SAND2000-2846
Unlimited Release
Printed November 2000

Development and Application of Genetic Algorithms For Sandia's RATLER Robotic Vehicles

Daniel W. Barnette
Parallel Computational Sciences Department

Richard J. Pryor
Evolutionary Computing Methods

John T. Feddema
Intelligent Systems Sensors & Controls

Sandia National Laboratories
PO Box 5800
Albuquerque, New Mexico 87185-0316

ABSTRACT

This report describes the development and application of genetic algorithms for the purpose of directing robotic vehicles to various signal sources. The use of such vehicles for surveillance and detection operations has become increasingly important in defense and humanitarian applications. The computationally parallel programming model as implemented on Sandia's parallel compute cluster Siberia and used to develop the genetic algorithm is discussed in detail. The model generates a computer program that, when loaded into a robotic vehicle's on-board computer, is designed to guide the robot to successfully accomplish its task. A significant finding is that a genetic algorithm derived for a simple, steady state signal source is robust enough to be applied to much more complex, time-varying signals. Also, algorithms for significantly noisy signals were found to be difficult to generate and should be the focus of future research. The methodology may be used for a genetic programming model to develop tracking behaviors for autonomous, micro-scale robotic vehicles.

Acknowledgments

The authors wish to acknowledge the following:

Funding for this project was obtained through Sandia's Laboratory Directed Research and Development (LDRD) office (Project 10690, Task 03).

Johnny Hurtado, Department 15211, supplied the mathematical models for the signal sources.

Wolfram Research's *Mathematica 4.0* was used to generate signal source and simulated robot path illustrations.

Contents

Development and Application of Genetic Algorithms For Sandia's RATLER Robotic Vehicles

Introduction

The emerging technical approach to deal with a challenging, possibly hostile, environment is likely to involve a large number of small, but fairly intelligent, robots. It is envisioned that these can covertly infiltrate a designated area, enter buildings, gather appropriate information, and communicate with and learn from each other. They would also communicate with a smaller number of on-the-scene soldiers backed up by powerful off-line computers that can carry out large-scale information collection, analyses, and simulations. Each robot would have on-board electronics, ground-positioning and communication equipment, an obstacle detector, and some source-analysis capability. Each robot would also have a motor, wheels, and a motor control system. Although the deployed robots would behave autonomously, each robot would communicate information with other robots during the task.

This report documents the effort to generate and apply a robust genetic algorithm to act as a vehicle controlling program for robotic behavior. In a typical scenario, robots are initially distributed randomly in a field and given the goal of locating the emitting source, be it sound or smell. An onboard processor running the algorithm provides instructions to the motor control system that directs the robot to the source location while navigating around obstacles.

The controlling algorithm is generated by a computer code designed to assemble, test, and compare many similar algorithms simultaneously. The code uses trial and error, tournament play, and best fits to generate a decision tree appropriate for the task. Once chosen, the decision tree then becomes the controlling algorithm of choice. The algorithm in decision-tree form is then translated into high-level computer language such as FORTRAN or C, compiled, and downloaded to the robotic vehicles deployed in the task. The robotic vehicles are then controlled by execution of the code using onboard processors, sensors, and memory.

The Sandia RATLER robotic vehicle serves as a research platform for the current effort. Although the RATLER's size precludes its large-scale use at present, further research will see the capabilities of RATLER reduced to micro-scale vehicles. Operationally, it is envisioned that tens to hundreds of these small robots would be deployed to complete a given task.

The Genetic Algorithm

Building a genetic algorithm is a compute-intensive process by virtue of the fact that it continually attempts to create successive generations of more fit algorithms. Improvement occurs in discrete steps called generations. A generation is composed of a population of individual algorithms each of which is a complete computer program. The number of algorithms considered at one time varies based on the problem; however, hundreds, if not thousands, of genetic algorithms can be considered simultaneously by judicious use of parallel computers. Typically, some algorithms will be more effective than others at doing the prescribed task. Each algorithm is scored for applicability, and its fitness is given a numerical score such that the higher the fitness, the better the algorithm. The goal is to generate an algorithm that correctly solves the problem of interest. However, there is no guarantee that the chosen algorithm is necessarily the best – usually, it suffices.

To create subsequent generations, genetic operators of selection, reproduction, crossover, and mutation are used. The purpose of selection is to choose an algorithm from the current population. In general, this algorithm will be better than most, but it may not be the very best. Reproduction moves a selected algorithm directly into the next generation. Crossover uses the selection operator twice to select two algorithms from the current population that will be combined in some way to form a hybrid algorithm that will be placed in the next generation. Mutation uses the selection operator once to choose an algorithm that will be changed in some way and placed in the next generation. The four genetic operators are discussed in more detail by Koza[1] and Pryor[2]. The development described above proceeds across many generations until a single algorithm is found that meets a convergence criteria. This algorithm is then tested and, if found to be sufficiently robust, implemented as the robotic controlling program.

Pryor[2] gives an example of the program representation of a decision tree making up a genetic algorithm. The basic building block of a tree is called a *node*, with all nodes in the tree having the same fixed structure. The first element of a node specifies the node type, which can either be a function or a terminal. A *function node* performs a mathematical or Boolean operation and generally has branches (nonzero pointers) that point to other nodes. The number of branches depends on the *kind* of function, e.g., add, subtract, multiply. A *terminal node* normally returns a value, does not have any branches (all pointers are zero), and terminates that section of the tree. Other elements within a node are a value position and pointers to other nodes. All of the nodes result in a decision tree that performs a specified task. More detail is given in the next section.

Noise can have a significant impact in actual applications where genetic algorithms are employed. Unless noise is accounted for during its creation, the genetic algorithm may not be able to respond in an appropriate manner. Such was the case in the present applications, as will be shown.

Table 1 lists functions allowed to make up the genetic algorithm.

Table 1: Functions and Terminals Available for Genetic Algorithm Decision Tree

No.	Function Name	Mathematical Expression	Comments
FUNCTIONS:			
1	RETURN		Returns a value
2	ADD	+	Adds two values
3	SUBTRACT	-	Subtracts two values
4	MULTIPLY	*	Multiplies two values
5	IFGTEQ	IF (value1 >= value2)	Compares two values
6	COMPUTE ANGLE		Determines which direction robot faces
7	STORE A-REG		Register for each robot
8	STORE B-REG		Register for each robot
9	STORE C-REG		Register for each robot
10	INTEGER ROUND	value=FLOOR(value1+0.5)	Round to nearest integer value
11	STORE AVG X-REG	$(1 - \kappa)$*(AVG X-REG) + κ*(AVG X-REG)	Exponential moving average for value stored (κ=0.5)
12	STORE AVG Y-REG	$(1 - \kappa)$*(AVG Y-REG) + κ*(AVG Y-REG)	Exponential moving average for value stored (κ=0.5)
TERMINALS:			
13	NEIGH 1 XPOS		First nearest neighbor's X location
14	NEIGH 2 XPOS		Second nearest neighbor's X location
15	NEIGH 1 YPOS		First nearest neighbor's Y location
16	NEIGH 2 YPOS		Second nearest neighbor's Y location
17	ROBUG XPOS		X location of robot
18	ROBUG YPOS		Y location of robot
19	ROBUG DIRECTION	1=North, 2=East, 3=South, 4=West	Direction heading of robot (1,2,3, or 4) on grid
20	NEIGH 1 SIGNAL		Signal detected by first nearest neighbor
21	NEIGH 2 SIGNAL		Signal detected by second nearest neighbor
22	ROBUG SIGNAL		Signal detected by robot
23	V-WALL XPOS		North-South wall's X location of corner
24	V-WALL YPOS		North-South wall's Y location of corner
25	H-WALL XPOS		East-West wall's X location of corner
26	H-WALL YPOS		East-West wall's Y location of corner
27	RECALL A-REG		Use A-register's contents
28	RECALL B-REG		Use B-register's contents
29	RECALL C-REG		Use C-register's contents
30	TURN NORTH		Directs robot to face North
31	TURN EAST		Directs robot to face East
32	TURN SOUTH		Directs robot to face South
33	TURN WEST		Directs robot to face West
34	TURN RIGHT		Directs robot to turn right
35	MOVE AHEAD		Directs robot to move ahead
36	VALUE		Store a value

Program Representation of the Genetic Algorithm

This section describes how the genetic algorithm is represented by individual program elements that make up a decision tree. The representation should always allow complete flexibility in defining programs, yet it must also ensure that the performance of the genetic operations is not too cumbersome. A tree-like structure best meets these requirements.

The basic building block of a tree is called a *nod*e, with all nodes in the tree having the same fixed structure. The first element of a node specifies the node type, which can either be a function or a terminal. A *function node* performs a mathematical or Boolean operation and generally has branches (nonzero pointers) that point to other nodes. The number of branches depends on the *kind* of function, e.g., add, subtract, multiply. A *terminal node* normally returns a value, does not have any branches (all pointers are zero), and terminates that section of the tree. Other elements within a node are a value position and pointers to other nodes.

Consider the sample decision tree shown in Fig. 1. This tree has five nodes and is three levels deep. The tree is evaluated by starting at its root, or top, and working downward until a terminal node is reached. A terminal node returns a value that is then processed upward in the tree.

To evaluate the sample tree, we begin at the first node denoted by **Start**, a function node, whose kind is specified as **Add**. This kind of function node points to two other nodes that return values to be summed by the **Add** node. At Pointer 1, there is a terminal node that returns a constant value of 2.3. At Pointer 2, there is a function node whose kind is **Multiply**. This node uses Pointers 3 and 4 to point to two **Value** nodes: one that returns the value 5.9, and the other the value of a global variable **x**. These two values are processed by the **Multiply** node, which returns the resultant along with the value of 2.3 to the **Add** node above it. The tree is equivalent to the expression

$$\boldsymbol{y} = \mathbf{2.3 + 5.9}\ \boldsymbol{x}$$

where $\boldsymbol{y}$ is the value returned by the root node at the top of the tree.

Larger trees used in the robotics program have many more function and terminal types than in the sample tree. The user specifies a maximum allowable number of nodes and depths in the code that generates the genetic algorithm, but typical values are around 800 nodes with a maximum tree depth of 80 levels.

Problem Setup

The computer code CEDAR has been written to assemble and test genetic algorithms using computer-simulated robots. A listing of pseudo-code for CEDAR is given in Appendix A. Using CEDAR's most current genetic algorithm, the computer-simulated robots solve a set of 90 problems to determine robustness and best fit for search-and-find behavior. At the start of each problem, the simulated robots are placed in arbitrary positions onto a two-dimensional grid and are tasked with finding an arbitrarily placed target. The option exists to arbitrarily place two walls onto the same grid to have the robots learn to avoid obstacles. The walls, if simulated, follow the grid lines in either the x or y directions. The goal for the simulated robots is to learn to navigate to the target and to avoid walls if present. The robots have no foreknowledge of either their own positions or the positions of the walls and the target.

Fig. 2 illustrates a representative configuration at startup. Select sequences of graphs show how the simulated robots converge on the two targets. Two signal sources, or targets, are shown in blue. Each signal represents a $1/r^2$ source whose strength is indicated by gray contours. Red circles randomly spaced about the blue targets represent the robots. In this application, walls were included in the simulation and are shown by heavy, intersecting lines. The position of each robot is given by a coordinate pair (x,y) which are positive integers. A robot's orientation can be in one of four directions, N, S, E, or W, where north is towards the top of the page. The direction impacts the robot's sensing ability: a robot is programmed to only sense an obstruction if it is positioned in the direction the robot is facing. As shown in the sequence, the simulated robots successfully avoid the walls and converge on the targets, *i.e.*, the signal peaks.

Certain assumptions related to actual robots are inherent in the problem setup. For example, it is assumed that memory on the robot's on-board computer is limited, and only four values of data are stored. Also, communication between robots is limited to the two nearest neighbors. The data that can be communicated consist of positions and signal strengths. Because of assumed limits in the motor control system, only one movement instruction can be returned with each execution of the behavior program. This instruction allows the robot to move ahead one grid point or to turn to a new direction while maintaining position.

CEDAR's computer simulations to generate a suitable genetic algorithm consisted of a population of 200 to 500 robots on each processor, running up to 128 processors on Sandia's compute cluster Siberia (http://www.cs.sandia.gov/cplant). Obstructive walls were sometimes included so that the simulated robots would learn to maneuver around obstacles. A photograph of Siberia, a cluster visually similar to an older cluster named Alaska, is presented in Fig. 3. Siberia and Alaska's configuration at the time of this writing is given in Table 3.

Table 2. Equations for Signal Source Models

Model #	Equation(s)
0	$1/r^2$
1	$-e^{-2r^2}[\cos(\pi + 20r + 5\theta - t) - 4]$
2	$-e^{-2r^2}[\cos\pi\cos(10r - 4t)\cos(5\theta - 4t) - 4]$
3	$-e^{(-\frac{x^2}{100}-\frac{y^2}{\sigma^2})}\frac{1}{\sigma\sqrt{2\pi}}[\cos\pi\cos(2x-4t)\cos(4y-4t)-4], \quad x \geq 0$ and $-e^{(-x^2-\frac{y^2}{\sigma^2})}\frac{1}{\sigma\sqrt{2\pi}}[\cos\pi\cos(2x-4t)\cos(4y-4t)-4], \quad x < 0$ where $r = \sqrt{x^2+y^2}$, $\theta = \arctan(\frac{y}{x})$, $t = \text{time}$, and $\sigma = x^2 + 2$

Signal Source Models

A total of four signal source models were considered. The equations governing each are given in Table 2. Mathematical representations of the models, generated using *Mathematica*[3], are shown in Fig. 4. The simplest model, Model 0, consists of a $1/r^2$ steady-state signal, as illustrated in Fig. 4a. Three time- and spatially-varying models were also considered. Illustrated in Figs. 4b, c, and d are Models 1, 2, and 3, respectively. Each of these functions has multiple local peaks that move around considerably as the robots search for the most likely signal peak. A sample of the maximum-peak movement for the unsteady models is given in Figs. 5a, b, and c.

Results

A. Simulation: Genetic Algorithm for Non-Noisy Signal Sources

The first attempts at generating genetic algorithms centered on modeling each signal source model. Wall-like obstructions were placed randomly on the grid so that the simulated robots would learn to maneuver around them. This became a very computationally intensive process since the signal sources for Models 1, 2, and 3 were time-varying and complicated, especially near the multiple center peaks.

Various attempts were made to accelerate the convergence. For example, more functions, such as exponential and sinusoidal, were added to the code from which the genetic algorithm would be generated. The reasoning behind this approach was that if the algorithm needed an exponential function that would otherwise be built from Taylor-series-like terms, for example, then adding this function to the list of possible functions to

Table 3. Configuration of Sandia's Parallel Compute Clusters
(go to http://www.cs.sandia.gov/cplant for more information).

	ALASKA	SIBERIA
Processor	DEC Alpha EV 56	DEC Alpha EV6
Processor speed	500 MHz	500 MHz
Operating system	Linux	Linux
Total number of nodes	270	592
Number of processors per node	1	1
Memory per node	196 MBytes	560 nodes have 256 MBytes; 32 nodes have 1 GByte
Parallel I/O bandwidth (scalable)	40 MBytes/sec over 4 network connections (enfs)	40 MBytes/sec over 4 network connections (enfs)

be chosen would negate the necessity to build the Taylor series. However, this also proved to be slowly convergent, possibly because the functions added too much complexity to the simpler algorithm being generated at the time. Also, adding more choices to the list of available functions algebraically increased the algorithm's number of options from which to choose as a decision tree was formed. That is, the function could be considered for use in each node in a tree. As a result, convergence to a best-fit algorithm became extremely tedious.

To alleviate the convergence problems, the authors decided to examine the possibilities of using the algorithm generated for the steady-state Model 0 for the time-varying models. No walls were simulated since it was decided that the to-be-conducted field tests would not initially contain obstructions.

Simulated vehicle movement using the genetic algorithm derived for Model 0 was generated using the code ROBOCOP, listed in Appendix B. The genetic algorithm generated by CEDAR is inserted in the function *MoveGA*, the last function listed in Appendix B, to complete the code. Sample output of ROBOCOP is presented in Appendix C. The *Mathematica* program used to graphically display the results is given in Appendix D.

The Model 0 result is illustrated in Fig. 6a, while the application of the identical algorithm to the remaining models is shown in Figs. 6b, c, and d. As shown, the algorithm worked very well for all signals for the sampling rates considered. This seems a surprising result considering the complexity of the other models compared to Model 0. However, further thought leads one to conclude that a single-peak-finding algorithm for steady-state signals may well be sufficient even for time-varying, multiple-peak signal sources as long as sampling rates are high relative to peak movement.

B. Field Test: Coupling the Genetic Algorithm with Sandia's RATLERs

The rovers used in the field tests were Sandia's Robotic All-Terrain Lunar Exploration Rover (RATLER) vehicles. Typical RATLER vehicles are shown in Fig. 7. The largest are approximately the size of two shoeboxes placed side by side. RATLER vehicles were developed by Sandia as a prototype vehicle for a lunar mission. Each vehicle is typically equipped with an Intel 486 computer, differential GPS receiver, spread spectrum two-way packet radio, electronic compass and tilt sensors, video camera, and RF video transmitter. Three RATLERs of the type shown in Fig. 7a were used during the tests. This was the minimum number needed for vehicle-to-vehicle communications as provided for in the genetic algorithm.

The base station equipment with which the RATLERs stay in constant communication consists of a Pentium laptop computer, spread spectrum two-way packet radio, differential GPS base receiver, RF video receiver, and a battery power source. The equipment is contained within a small trailer for mobility. The base station sends commands and queries to the RATLERs over the packet radio. The communication network is configured as a token ring. Hence, if the base station becomes non-functional, the vehicles will continue to communicate. Also, if either the vehicle or base station misses its turn to communicate, communications can be re-established after a specified delay.

During field tests, the operator places the RATLERs in autonomous navigation mode. A live video image from one of the vehicles can be displayed on the laptop along with the current position of the vehicle on a Geographic Information System (GIS) map. Multiple RATLERs are driven to operator-specified set points using differential GPS and a magnetic compass, where they are allowed to navigate on their own to the source using the genetic algorithm controlling program. The positioning accuracy of the vehicles is typically 1 meter.

As a result of its success in finding the peaks of all signal models, the algorithm for Model 0 was implemented on robotic rovers for field tests. The signal source was a loud speaker placed in a large field so as to closely simulate the $1/r^2$ Model 0 source. The RATLERs were placed in a random position about the source. The genetic algorithm previously loaded into the RATLERs onboard memory was then executed and the vehicles were allowed to move about as directed by the algorithm. No obstructions were placed between the rovers and signal source.

Direct observations of the ensuing test were that the vehicles found the source but wandered significantly beforehand. The wandering was attributed to signal noise that may have been caused by nearby vehicle traffic, wind, and possibly electronic component tolerances. Signal noise was not due to vehicle movement since signals were generated only after each RATLER stopped momentarily. Base station equipment recorded the noise to be as high as 10% of the signal source. Noise was not modeled in the initial algorithm for Model 0.

It was also discovered that a RATLER acting alone showed nearly identical behavior to that when all three were attempting to locate the target. This apparently indicated that the vehicles were not communicating with each other even though provisions such as registers were made available with the functions given in Table 1. Computer simulations using only one simulated robot reinforced the conclusion that the vehicles were not communicating as originally believed. Implications are that the vehicles were acting autonomously and not collectively, as should be in the case of a swarm of vehicles. It is unknown why this occurred, but it is certainly an area for future research.

C. Simulation: Non-Noisy Genetic Algorithm Applied to Noisy Signals

Noise was introduced into the signal source used in Model 0's simulator. The original genetic algorithm was then used to perform a post-mortem simulation and analysis of the field tests. A random number generator was used to perturb the original signal within a user-specified percentage at each time step. This would hopefully reveal the effects of noise on the robot convergence path.

Results of the robot convergence path are illustrated in Fig. 8. Shown in Fig. 8a is the convergence path for a 5% noise signal. The simulated rover finds the peak even though the signal strength is slightly perturbed. However, an 8% perturbation causes the simulated rover to never converge, as illustrated in Fig. 8b. Thus, the genetic algorithm for Model 0 is apparently robust enough to handle small perturbations to around 5%. However, higher perturbations cause the robot to wander as observed in the field test.

At this point, two alternatives became obvious to alleviate the wandering. The first approach was to lower the signal noise in some way. This was quickly abandoned, since the factors causing the noise levels were out of the operators' control. It is also possible that higher noise levels may exist during actual future applications and that these levels could not be predicted beforehand.

The second approach was to introduce noise in the code that generates the genetic algorithm.

D. Simulation: Genetic Algorithm for Noisy Signal Sources

It was thought that the genetic algorithm for Model 0 could be regenerated with the ability to process noisy signals. The new algorithm, if successful, would allow the rovers to find valid signal peaks even through a 'dirty' signal. Once again, so as not to add more complexity to the problem, walls were not modeled.

Significant computer time was spent on this approach, but without much success. Project deadlines prevented a thorough attack on this problem, but preliminary analyses indicated simulated rovers would find their way to a fairly large distance from the peak, and no closer. It is not entirely understood why this happened. However, better convergence might be achieved if the algorithm ensured each rover communicated with some of its nearest neighbors, thereby triangulating the signal source. As has been discussed, rover-to-rover communications were apparently not occurring in the original Model 0 genetic algorithm. Hence, another area of research should include a study of the ability of genetic algorithms to intelligently process noisy signals.

Summary and Conclusions

This report documents a research effort in which a genetic algorithm code was developed and ported to Sandia's parallel compute clusters. The code was modified to use the MPI message passing protocol. Efficiency was improved by reducing excessive message passing between the master node and slave nodes. The ability to investigate time-varying signal sources was added to the original code. Visualization schemes were developed and implemented for investigating simulated robot behavior before running field tests with actual hardware.

The result of this effort, a genetic algorithm, has been implemented in hardware as a robot controlling program. Field tests were conducted using Sandia's RATLER robotic vehicles attempting to locate a low humming stereo speaker. Tests were successful, though significant wandering was observed that was not evident during computer simulations. This behavior is believed to be due to signal noise. Project deadlines prevented generating a genetic algorithm that could filter noise and locate the peak efficiently. It was also noticed that the algorithm resulted in autonomous, rather than collective, robot behavior. The factors that govern this behavior should be a topic of future research.

An interesting finding of this research was the fact that a genetic algorithm developed for a simple test case proved very robust for more complex applications and signals. Computer simulations showed that the algorithm developed for a simple $1/r^2$ case proved sufficient for much more complicated applications. This should be kept in mind in any future research involving applying genetic algorithms to complicated applications: keep it as simple as possible. Extensions of simple algorithms may be possible for much more complicated applications.

In conclusion, the authors believe genetic algorithms have a strong future at Sandia, especially when applied to problems that have no definitive analytical answers, but where a 'good' solution will do. Future areas of research should include an approach that ensures rover-to-rover communications and the study of the effects of noisy signals on obtaining acceptable rover behavior. It is hoped that this report gives impetus to additional research in these areas so that more robust genetic algorithms may be developed.

References

1. Koza, J. R., Genetic Programming, On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

2. Pryor, R. J., "Developing Robotic Behavior Using a Genetic Programming Model," SAND98-0074, Sandia National Laboratories, Albuquerque, New Mexico, January 1998.

3. Wolfram, S., The Mathematica Book, 3rd edition, Wolfram Media & Cambridge University Press, 1996.

[Intentionally Left Blank]

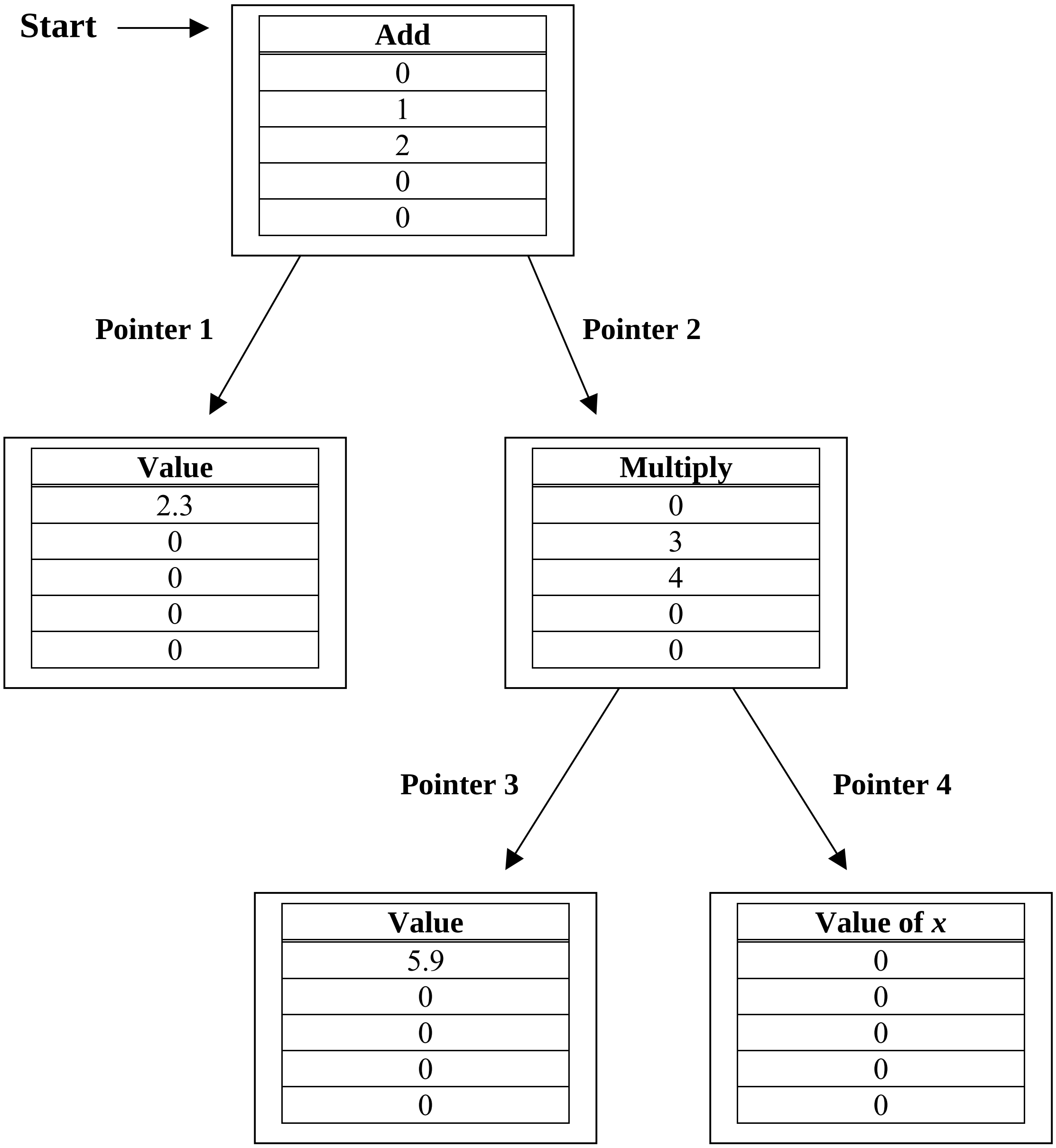
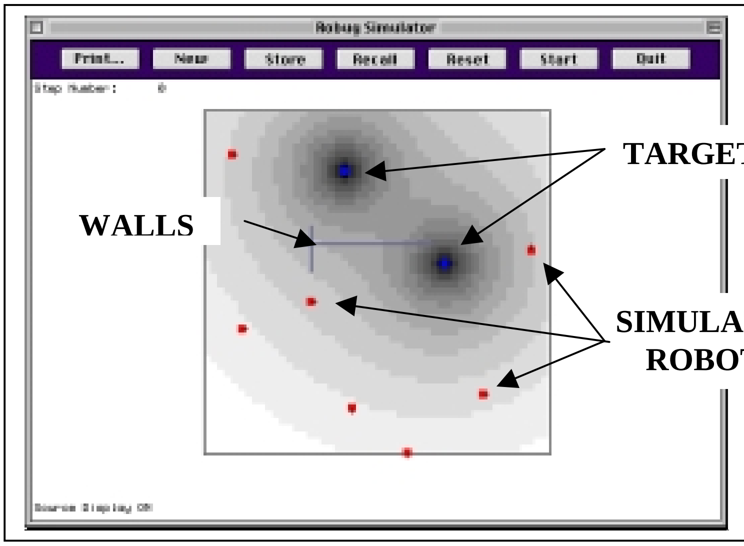
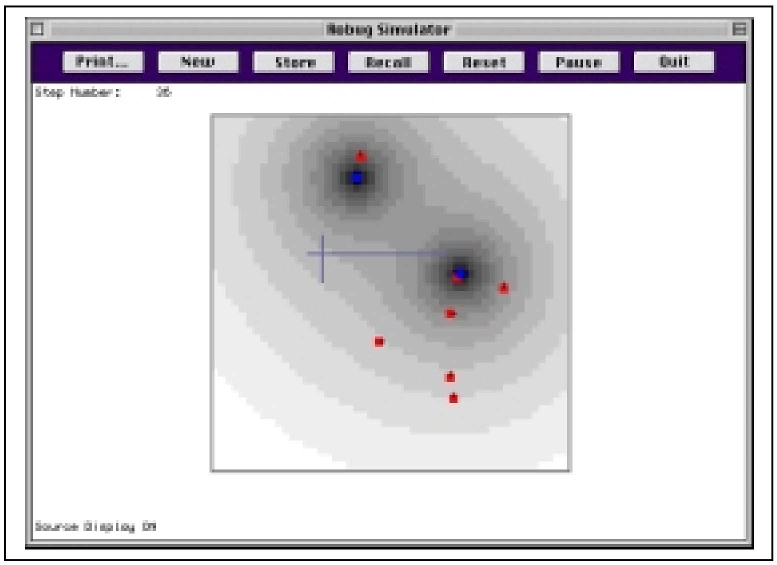


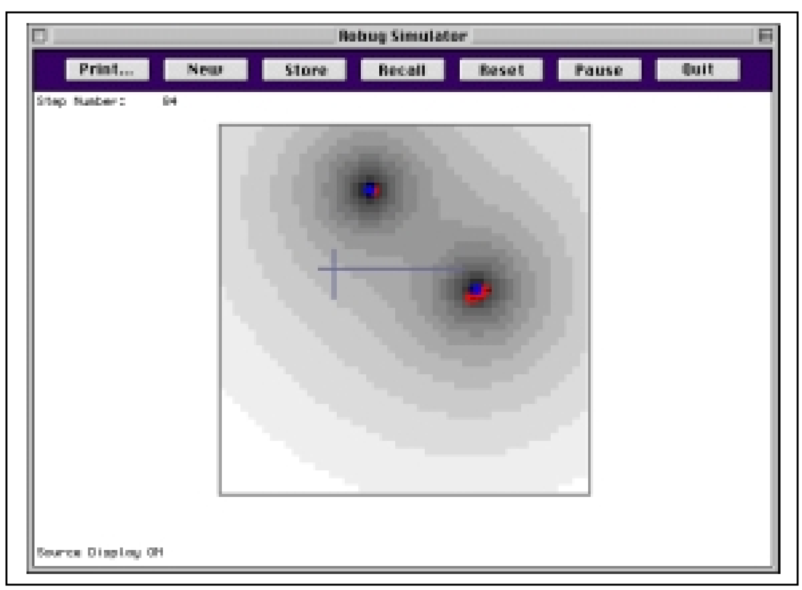
Figure 1. Example of a 5-node, 3-level decision tree representing $y=2.3 + 5.9x$.



a) Step 0



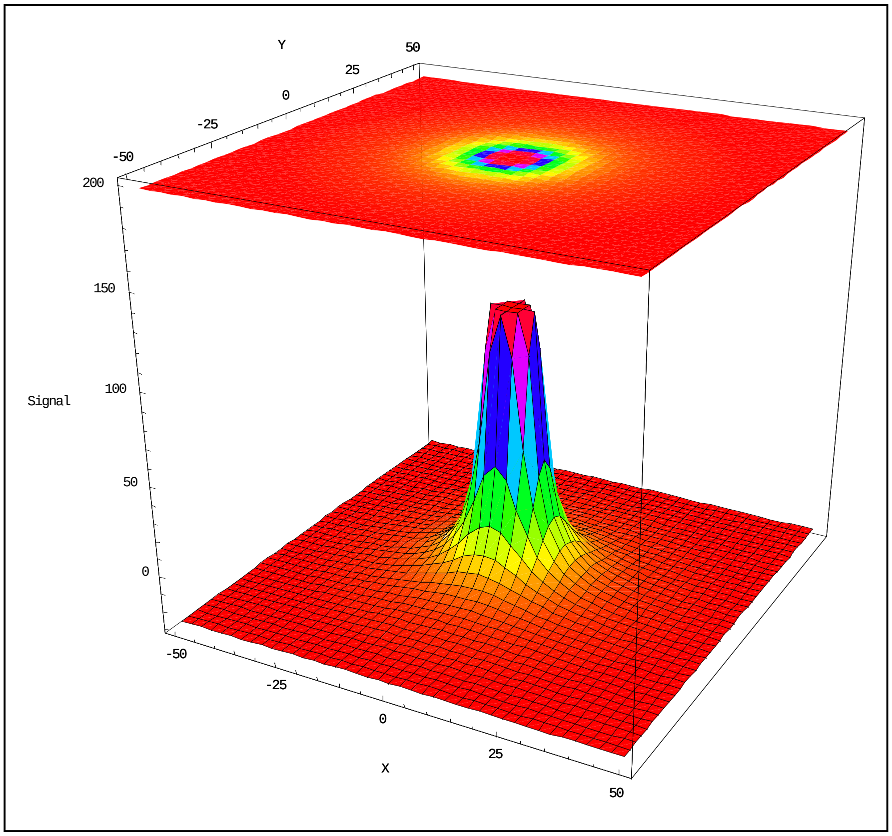
b) Step 36



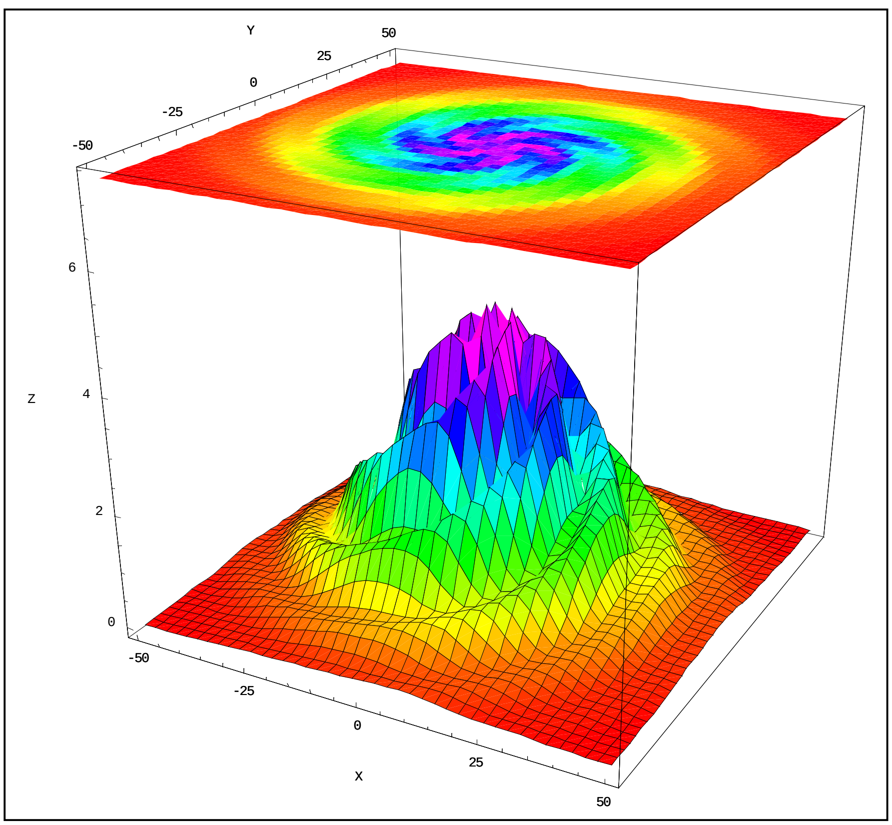
c) Step 84

Figure 2. Select, top-view convergence sequences of a representative genetic algorithm.

Figure 3. Sandia's parallel compute cluster, ALASKA/SIBERIA. For more details, go to web site http://www.cs.sandia.gov/cplant.

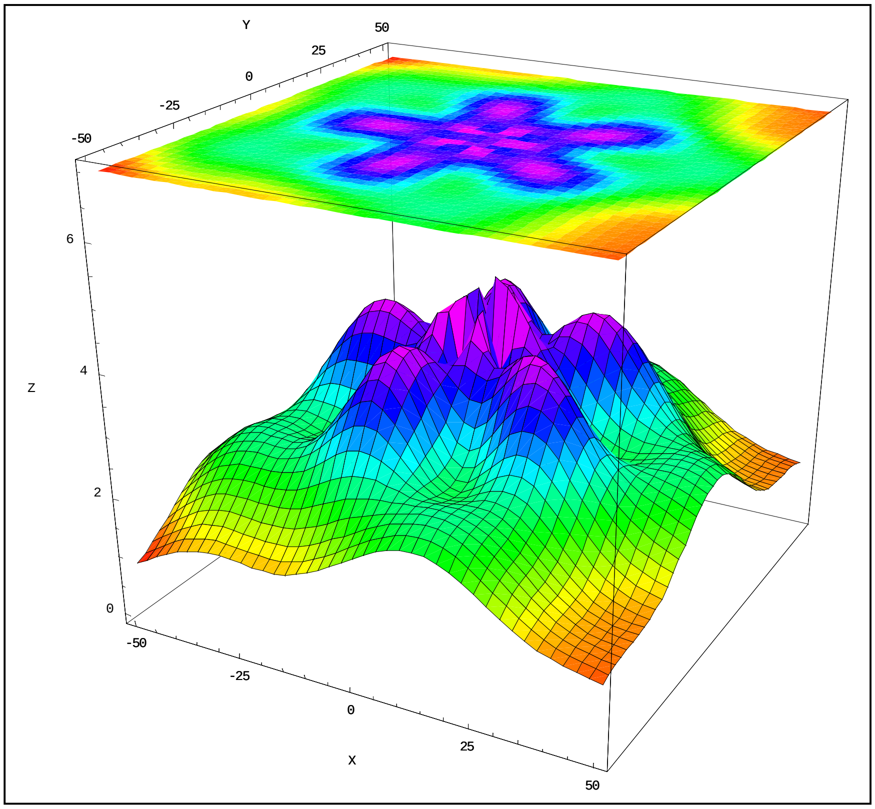


a) Model 0

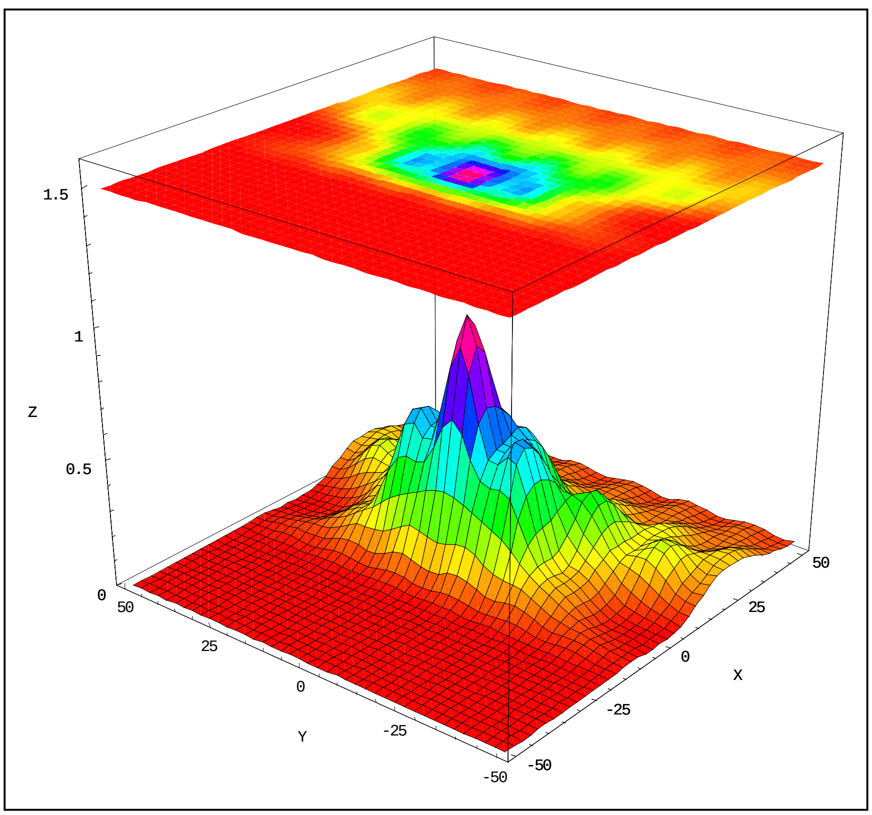


b) Model 1

Figure 4. Various source signal models used.

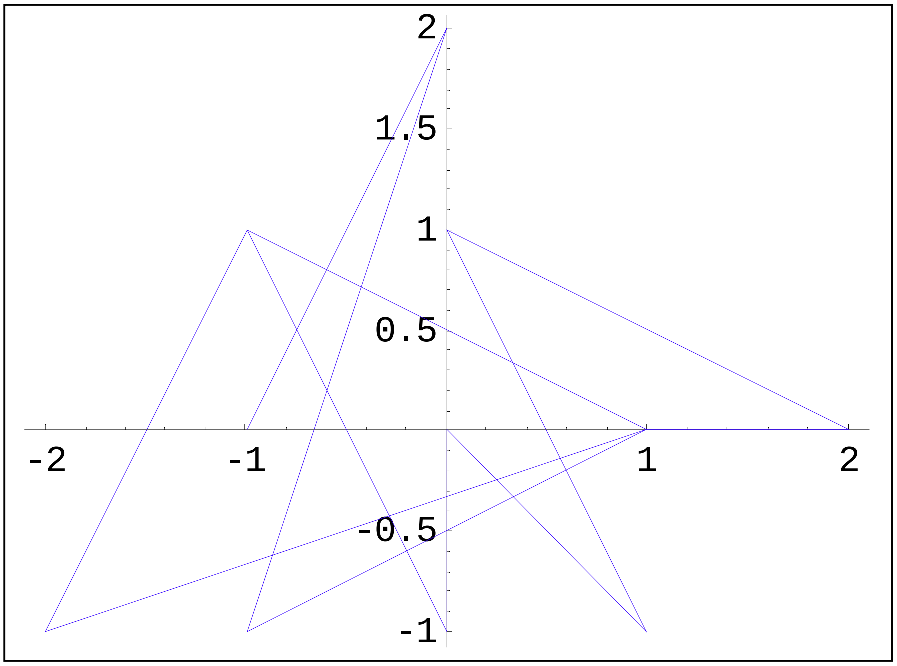


c) Model 2

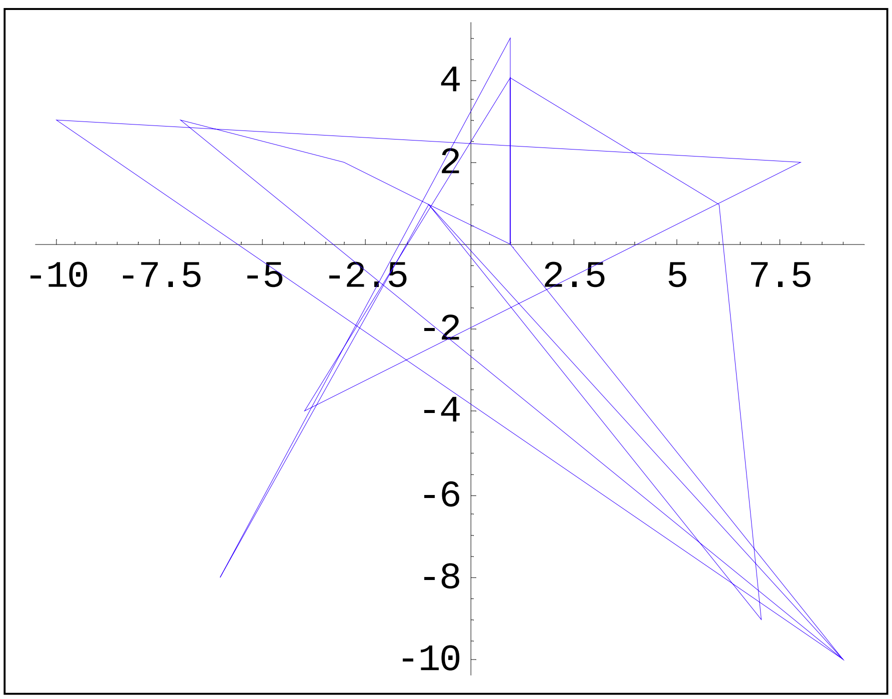


d) Model 3

Figure 4. Concluded

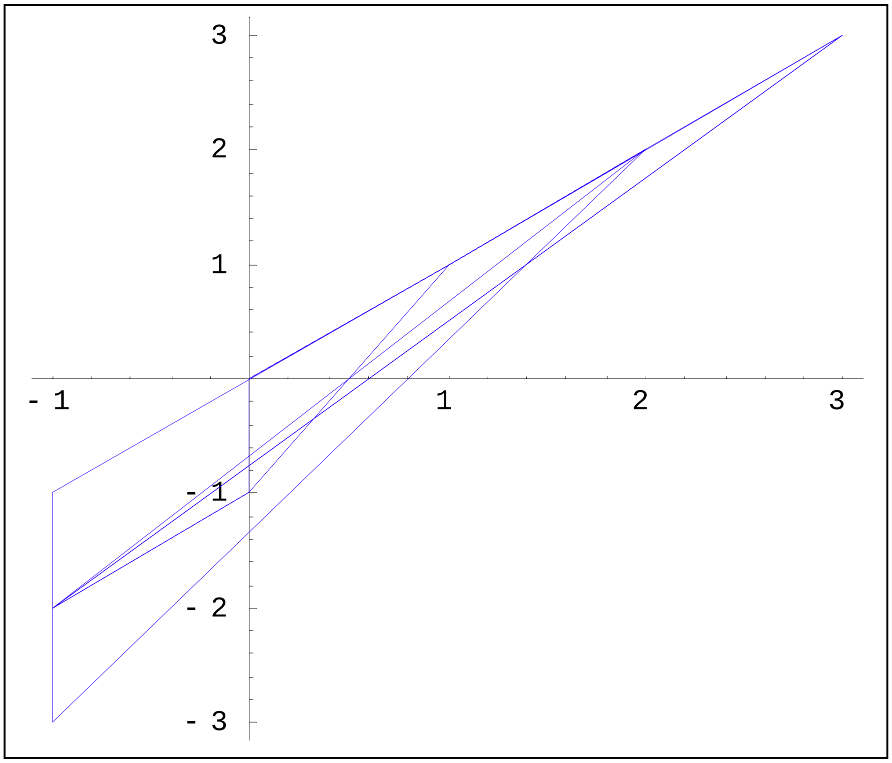


a) Model 1, 0<t<10 *sec*



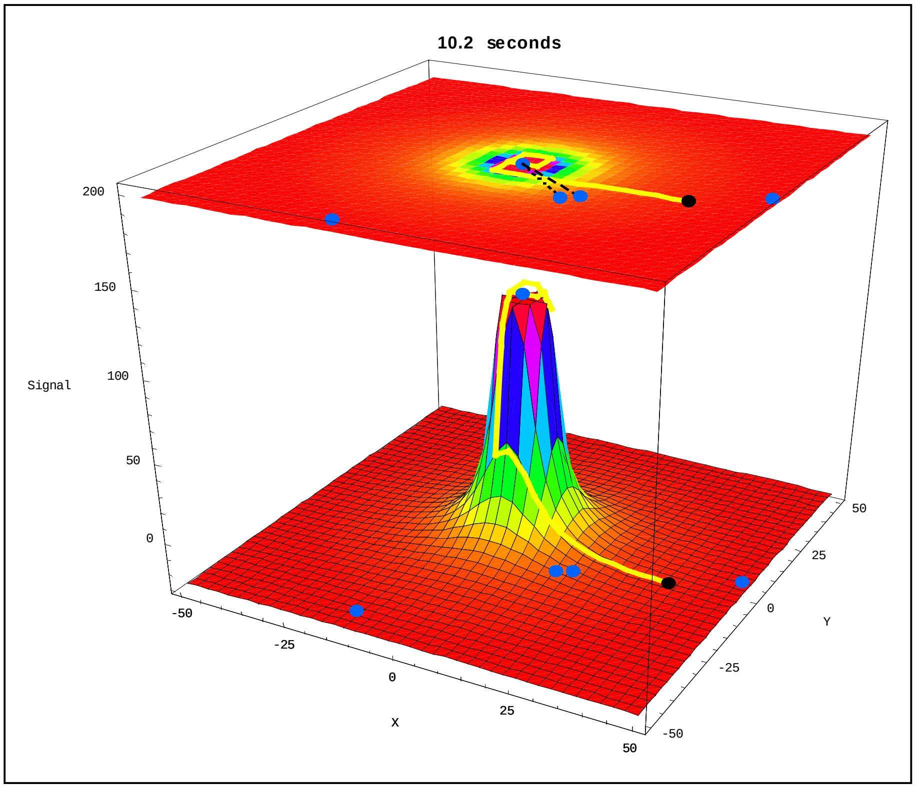
b) Model 2, 0<t<10 *sec*

Figure 5. Peak movement for the various signal source models.

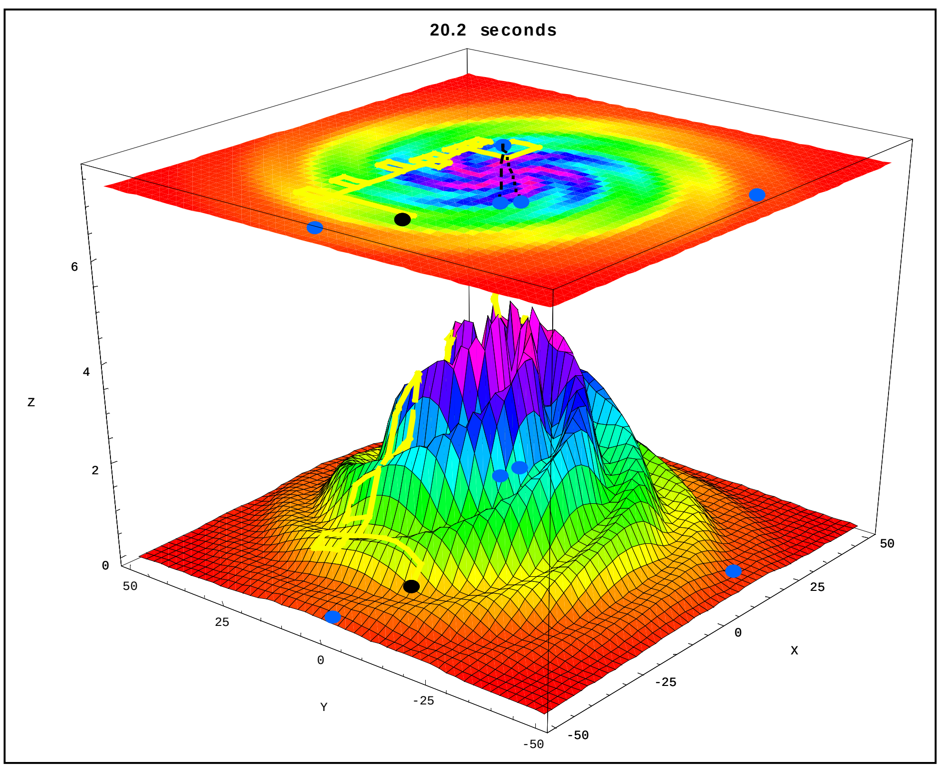


c) Model 3, 0<t<10 *sec*

Figure 5. Concluded.

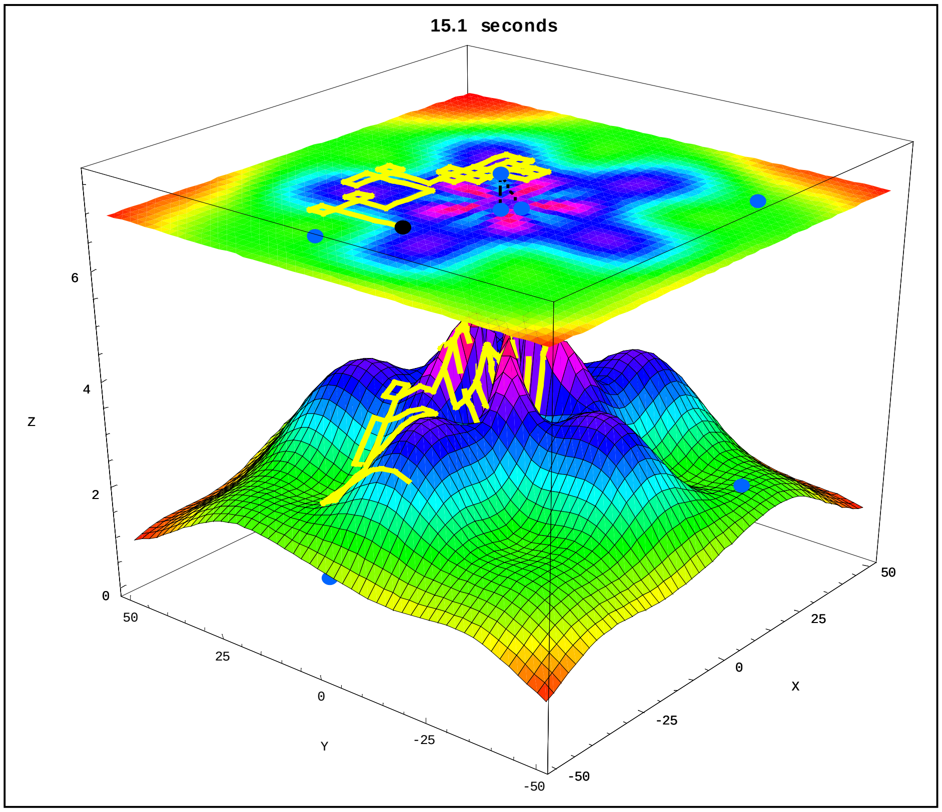


a) Model 0, t=10.2 *sec*

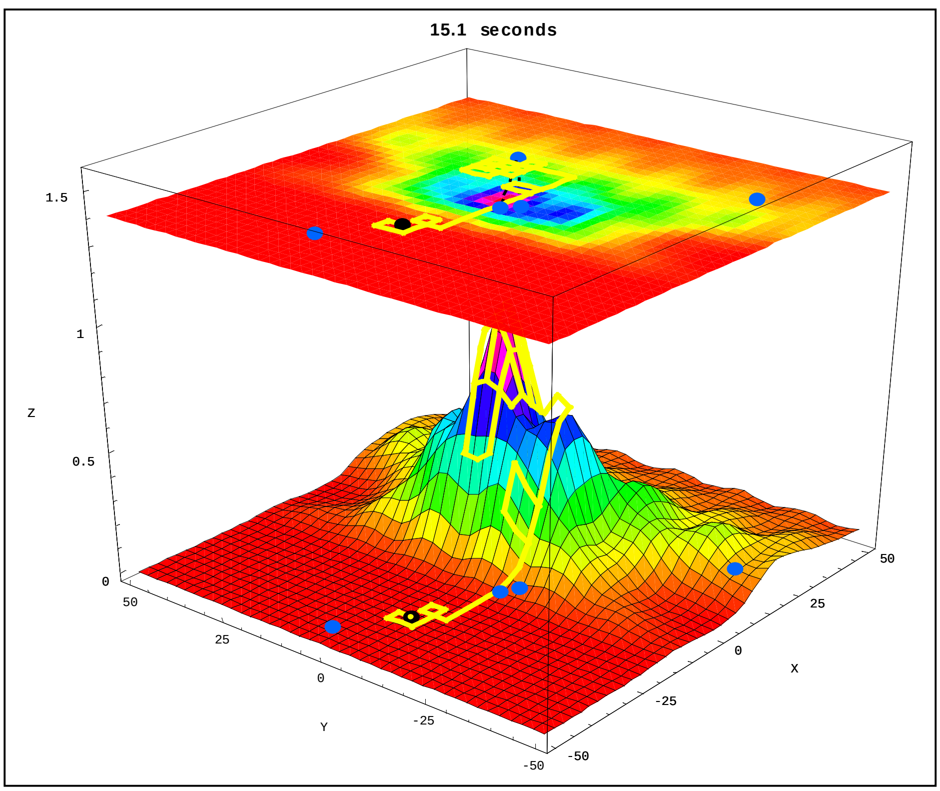


b) Model 1, t=20.2 *sec*

Figure 6. Simulated vehicle movement for the signal source models.



c) Model 2, t=16.1 *sec*



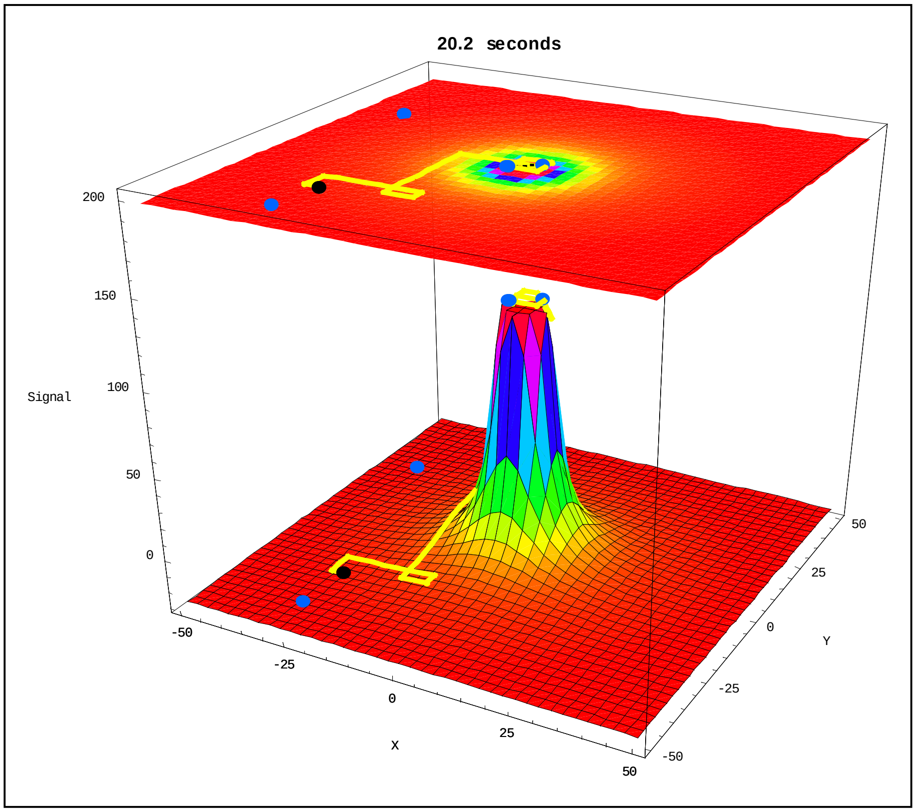
d) Model 3, t=15.1 *sec*

Figure 6. Concluded.

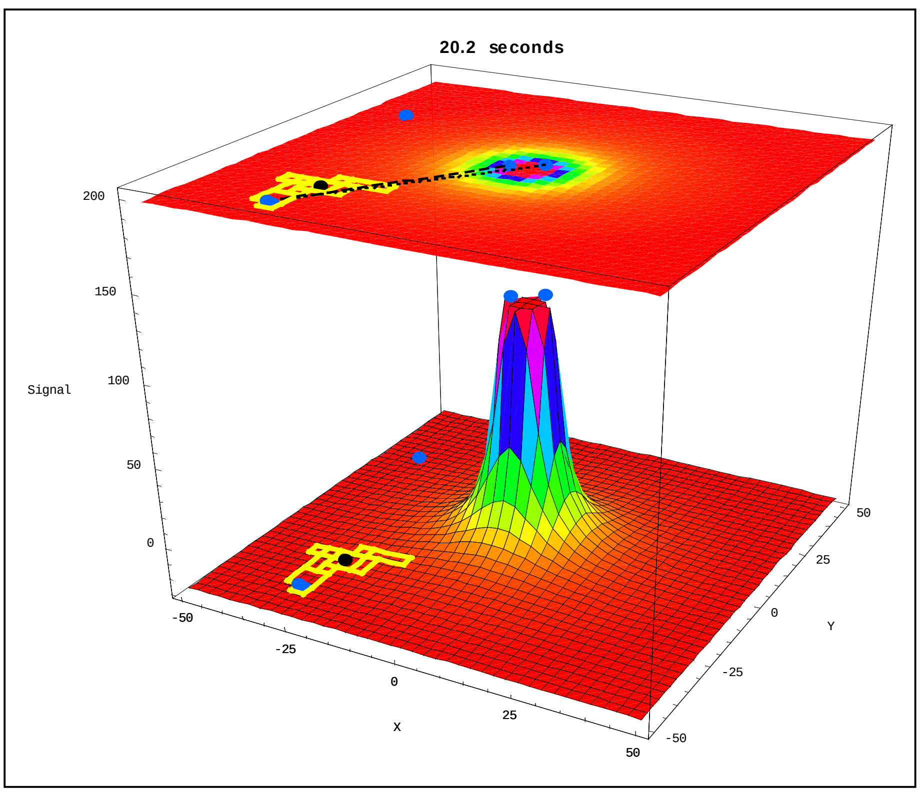
a)

b)

Figure 7. RATLER vehicles developed at Sandia National Laboratories.



a) 5% signal noise



b) 8% signal noise

Figure 8. Model 0 source illustrating algorithmic convergence for 5% signal noise and non-convergence for 8% signal noise.

[Intentionally Left Blank]

Appendix A
CEDAR Pseudo-Code Program Listing

	> # include Header files
	> Define global variables
	> #define Constants and parameters
1)	**main (unsigned int argc, char *argv[]) {** ➢ Initialize Message Passing Interface (MPI) environment ➢ if (NodeNum == 0) ➢ HandleNodeZero(); ➢ else ➢ HandleOtherNodes(); }
2)	**void HandleNodeZero () {** ➢ Print initialized quantities ➢ Call ReadList{} to read the best tree, if output from previous run ➢ Call randomng{} to generate random number ➢ Loop over the number of generations: ➢ Loop over number of compute nodes ➢ Wait for incoming trees to node zero ➢ Check each incoming tree for fitness; if best, replace previous tree ➢ If maximum number of generations reached, send message to all nodes to quit ➢ Write the best tree to disk ➢ Return to main }
3)	**void HandleOtherNodes () {** ➢ Initialize trees, parameters ➢ Call randomng{} to generate random number ➢ Loop over population size o Determine and store best tree on current compute node ➢ Loop over number of generations o Implement mutation, reproduction, crossover to generate new tree o Define current best tree ➢ Send best tree to node zero ➢ Quit searching for best tree when node 0 says maximum number of generations reached ➢ Return to main }
4)	**double EvalTree (AtomType *ptr, long ntsx) {** ➢ Loop over number of tests to run to determine best trees o Define wall positions if used o Define initial target position; initial robot positions ➢ Loop over number of cycles to be done for each test ➢ Loop over number of robots

	o **Determine signal strength measured by each robot** o **Determine each robots' nearest robot neighbors** o **Move the robots according to current genetic algorithm** o **End loop over cycles** ➢ **Determine the distance the robots are from the target** ➢ **End loop over tests** ➢ **Determine the fitness of each tree** ➢ **Return the fitness value** **}**
5)	**AtomType * GetBestTree (AtomType *bestIndiv) {** ➢ **Receive the best tree from compute Node 0** **}**
6)	**void KillNodes () {** ➢ **Kill all compute nodes when finished** **}**
7)	**OSErr InitManager () {** ➢ **Initialize dynamic memory allocation** **}**
8)	**AtomType * myMalloc (void) {** ➢ **Initialize memory allocated for trees** **}**
9)	**void myFree(AtomType *ptr) {** ➢ **point to next location of free memory block** **}**
10)	**void HandleError (int errAction, OSErr err, char *script) {** ➢ **Error handling routine for CEDAR** **}**
11)	**double RandomDouble (double start, double stop) {** ➢ **Determine random number to type double precision** **}**
12)	**long RandomLong (long istart, long istop) {** ➢ **Determine random number to type long** **}**
13)	**double randomng(int *pp) {** ➢ **Random number generator** **}**
14)	**double eval (AtomType *ptr, RoBugType *bug) {** ➢ **Compute the value of each tree node using assigned functionals** **}**
15)	**long TreeSize (AtomType *ptr) {** ➢ **Compute the number of nodes in each tree structure** **}**

16)	**void CountNode (AtomType *ptr) {** ➢ **Count and set points for each node** **}**
17)	**long TreeDepth (AtomType *ptr) {** ➢ **Determine the number of levels of each tree** **}**
18)	**void NodeDepth (AtomType *ptr) {** ➢ **Determine depth of each tree node** **}**
19)	**void DeleteOffspring (AtomType *ptr) {** ➢ **Free memory from tree nodes no longer needed** **}**
20)	**AtomType * DuplicateTree (AtomType *fromptr) {** ➢ **Duplicate a tree node using CopyNode** **}**
21)	**void CopyNode (AtomType *fromptr, AtomType *ptr) {** ➢ **Free a block of memory** ➢ **Copy a tree node from one block of memory to another** **}**
22)	**AtomType * CrossOver (AtomType *ptr1, AtomType *ptr2) {** ➢ **Perform crossover algorithm on two tree nodes' offspring** **}**
23)	**void PrintTree (AtomType *ptr) {** ➢ **Print size and depth of tree to appropriate output file** **}**
24)	**void PrintNode (AtomType *ptr) {** ➢ **Print tree nodes** **}**
25)	**AtomType * GenerateTree () {** ➢ **Initialize maximum depth of initial tree** **}**
26)	**void GenNode (AtomType *ptr) {** ➢ **Randomly generate tree nodes** **}**
27)	**void WriteCProgram (AtomType *ptr, long progIndex) {** ➢ **Output initialization information formatted in C** **}**
28)	**void ArrayNode (AtomType *ptr, long nsize) {** ➢ **determine memory pointers for tree nodes** **}**

29)	**void WriteCProgramNode (AtomType *ptr, FILE *ProgramFP) {** ➢ **Output tree nodes formatted in C** **}**
30)	**void WriteFProgram (AtomType *ptr, long progIndex) {** ➢ **Output initialization information formatted in FORTRAN** **}**
31)	**void WriteFProgramNode (AtomType *ptr, FILE *ProgramFP) {** ➢ **Output tree nodes formatted in FORTRAN** **}**
32)	**long SelectTree (double fitness[]) {** ➢ **Determine which tree is best** **}**
33)	**AtomType * TreeToList (AtomType *ptr) {** ➢ **Determine absolute to relative memory address for each node** **}**
34)	**void ListNode (AtomType *ptr) {** ➢ **Determine pointer values for each tree's nodes** **}**
35)	**void AbsToRelAdr(AtomType *ptr, long size) {** ➢ **Determine node's relative address, given absolute address** **}**
36)	**void RelToAbsAdr (AtomType *ptr, long size) {** ➢ **Determine node's absolute address, given relative address** **}**
37)	**void WriteList (AtomType *ptr, long listIndex) {** ➢ **Open and write list file** **}**
38)	**AtomType * ReadList (char ListFileName[]) {** ➢ **Open and read list file** **}**
39)	**void MutateTree (AtomType *ptr) {** ➢ **Determine pointer of tree node at which to begin mutation** ➢ **Determine level at which tree is mutated** ➢ **Generate new tree starting at mutation point** **}**
40)	**void ZeroHitCount (AtomType *ptr) {** ➢ **Zero node hit count** **}**
41)	**void ClearGrid (void) {** ➢ **Sets all points in the grid table to 0** **}**

42)	**void ClearGridPoint (long xpoint, long ypoint) {** ➢ **Sets individual grid points in the grid table to 0** **}**
43)	**long SetGridPoint (long xpoint, long ypoint) {** ➢ **Check fidelity of grid points to ensure grid points are within specified ranges** ➢ **Check that grid points to which robots can potentially move are not a wall point** **}**

==================================

Appendix B
ROBOCOP.C Program Listing
(as run for Model 0)

```
/*************************************************/
/*                                               */
/*  The GA Robug Simulator Program.              */
/*    This is a test program before implementing */
/*    GA code on hardware                        */
/*                                               */
/*  Last modified on:  April, 2000               */
/*                                               */
/*  Developed by:                                */
/*              D. Barnette                      */
/*              J. Feddema                       */
/*                                               */
/*************************************************/

/********************************/
/*      header files            */
/********************************/

#include "sys/stat.h"
#include "sys/types.h"
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "signal.h"
#include "math.h"

/********************************/
/*      output files            */
/********************************/
/* Home directory (use \\ for each \ needed) */
char        HomeDirectory[]="D:\\Program Files\\DevStudio\\MyProjects\\Robug_Simulator\\";
FILE        *OutputFP;
char        OutputFileName[50];

/********************************/
/*       defines         */
/********************************/

#define grid_dim_lat            11          /* number of latitude points */
#define grid_dim_lon            11          /* number of longitude points */
#define  grid_spacing_lat_m     3           /* meters (integer values, >= 1); spacing between
latitude grid points  */
#define grid_spacing_lon_m      3           /* meters (integer values, >= 1); spacing between
longitude grid points */
#define  grid_scale_delta_lat_mpmas        0.0310444444
#define grid_scale_delta_lon_mpmas         0.0254302890
                        /*  2.54303 cm = milliarcsecond latitude */
                        /*  3.10444   cm = milliarcsecond longitude */
                         /* 2543.03 cm = 1 arcsec latitude  = 25.4321 meters  */
                         /* 3104.44 cm = 1 arcsec longitude = 31.0424 meters */
                         /*  0.0254303 m = 1 milliarcsecond latitude  */
```

```
                              /*  0.0310444 m = 1 milliarcsecond longitude */
                              /*  3 meters = 117.9612 milliarcseconds latitude */
                              /*  3 meters =  96.6420 milliarcseconds longitude */

#define midpoint_lat                (grid_dim_lat+1)/2
#define midpoint_lon                (grid_dim_lon+1)/2

#define numvehicles                     5     /* total number of vehicles */
#define this_vehicle_ID                 3     /* choose number between 0 and (numvehicles-1) */
#define REALLYBIGNUMBER         99999999999999.0 /* arbitrarily-big number */
#define TEST2SEED                       765  /* random number generator seed */
#define SignalStrengthMax             4095 /* robot hardware sees 0-4095 signal strength */
#define GridMaxRadius                   50    /* plus and minus values in meters in which robots are placed
                                                                 relative to source */

/* define target location */
#define target_location_lat 126000000        /* Lat =  35 degs North in milliarcseconds*/
#define target_location_lon 921600000        /* Lon = 256 degs East  */

#define total_iterations                 50    /* run this many iterations */
#define iterations_per_second            5     /* iterations per second */
/* Note: total time = total_iterations / iterations_per_second */

#define Pi                               3.141592654


/********************************/
/*          structs             */
/********************************/
typedef struct RoBugLocation
{
   long    RoBugID;
   long    RoBugXPOS;
   long    RoBugYPOS;
   long    RoBugDIR;
   double  RoBugSIG;
   long    RoBugXVERT;
   long    RoBugYVERT;
   long    RoBugXHORZ;
   long    RoBugYHORZ;
   long    RoBug1XPOS;
   long    RoBug1YPOS;
   double  RoBug1SIG;
   long    RoBug2XPOS;
   long    RoBug2YPOS;
   double  RoBug2SIG;
   double  aRegister;
   double  bRegister;
   double  cRegister;
          double   Average_X_Register;
          double  Average_Y_Register;
          long      WallAhead;
} RoBugType;


/********************************/
/*          globals             */
/********************************/
```

```
float           time;
double          delta_lat, delta_lon;
int              iseed;
double          sseed;

/* static memory allocation */
/* RoBugType     RoBug[numvehicles]; */

/* dynamic memory allocation */
RoBugType       *RoBug;

// In InitGA(), RoBug = (RoBugType *)malloc( numvehicles * sizeof( RoBugType ) );
// Called RoBug[i] in other routines.


/*********************************/
/*              routines             */
/*********************************/
int             SignalOne( long lat, long lon );
void            TestGA();
void            InitGA();
void            UpdateGA(long *dlat, long *dlon, int id, long lat, long lon, int amplitude );
void            KillGA();
void            Update_VehicleGA(int id, long lat, long lon, int amplitude);
double          MoveGA(RoBugType *bug);
double          RandomDouble ( double start, double stop );
long            RandomLong (long istart, long istop);
double          randomng(int *pp);
double          MoveAhead (bug);
double          TurnRight (bug);
double          TurnNorth (bug);
double          TurnEast  (bug);
double          TurnSouth (bug);
double          TurnWest  (bug);


// Suggestion put main() and TestGA() in another file

/**************************** main ****************/
void main()
{
 /* define output file name */
   sprintf(OutputFileName,"%sRobocopOutputMod0.txt",HomeDirectory);

/*   sprintf(OutputFileName,"GPOutput"); */

/* open output data file */
   OutputFP=fopen(OutputFileName,"w");

  iseed=TEST2SEED;

 delta_lat = grid_spacing_lat_m / grid_scale_delta_lat_mpmas; // milli-arcseconds per grid point
 delta_lon = grid_spacing_lon_m / grid_scale_delta_lon_mpmas; // milli-arcseconds per grid point

 /* to console */
 printf(" total iterations = %d\n",total_iterations);
 printf(" iterations per second = %d\n",iterations_per_second);
```

```
 printf(" number of vehicles = %d\n",numvehicles);
 printf(" grid_spacing_lat_m = %d\n",grid_spacing_lat_m);
 printf(" grid_spacing_lon_m = %d\n",grid_spacing_lon_m);
 printf(" grid_scale_delta_lat_mpmas = %f\n",grid_scale_delta_lat_mpmas);
 printf(" grid_scale_delta_lon_mpmas = %f\n",grid_scale_delta_lon_mpmas);
 printf(" delta_lat (mas/lat_grid_step) = %f\n",delta_lat);
 printf(" delta_lon (mas/lon_grid_step) = %f\n\n",delta_lon);

 /* to OutputFileName */
 fprintf(OutputFP," total iterations = %d\n",total_iterations);
 fprintf(OutputFP," iterations per second = %d\n",iterations_per_second);
 fprintf(OutputFP," number of vehicles = %d\n",numvehicles);
 fprintf(OutputFP," grid_spacing_lat_m = %d\n",grid_spacing_lat_m);
 fprintf(OutputFP," grid_spacing_lon_m = %d\n",grid_spacing_lon_m);
 fprintf(OutputFP," grid_scale_delta_lat_mpmas = %f\n",grid_scale_delta_lat_mpmas);
 fprintf(OutputFP," grid_scale_delta_lon_mpmas = %f\n",grid_scale_delta_lon_mpmas);
 fprintf(OutputFP," delta_lat (mas/lat_grid_step) = %f\n",delta_lat);
 fprintf(OutputFP," delta_lon (mas/lon_grid_step) = %f\n\n",delta_lon);

        TestGA();
}


/**************************** TestGA ***************/
void TestGA()
{
        unsigned int timestep;
        int vehicle;
        long lat, lon, dlat, dlon;
        int amplitude, neighbor1_ID, neighbor2_ID;

        InitGA( );

/* check if the value 'this_vehicle_ID' is out of range */
        if(this_vehicle_ID >= numvehicles || this_vehicle_ID < 0 )
        {
                printf("\n Quantity this_vehicle_ID out of range \n");
                printf("     numvehicles = %d\n",numvehicles);
                printf(" this_vehicle_ID = %d\n",this_vehicle_ID);
                exit(1);
        }

/* print heading for output */
        printf("\n \n >> Locations of all vehicles except vehicle # %d \n",this_vehicle_ID);
        printf("  Time  Veh   Dir     Lat        Lon   Target_lat Target_lon"
                "    Signal  X dist(m) Y dist(m)\n");
        fprintf(OutputFP,"\n \n >> Locations of all vehicles except vehicle # %d \n",this_vehicle_ID);
        fprintf(OutputFP,"  Time  Veh   Dir     Lat        Lon   Target_lat Target_lon"
                "    Signal  X dist(m) Y dist(m)\n");

/* For other stationary vehicles */
        for(vehicle=0; vehicle < numvehicles; vehicle++)
        {

                if(vehicle != this_vehicle_ID)
                {
/* locate vehicles within + or - GridMaxRadius meters of specified target location */
```

```
        lat=target_location_lat + RandomDouble(-GridMaxRadius,GridMaxRadius) /
grid_scale_delta_lat_mpmas;
        lon=target_location_lon + RandomDouble(-GridMaxRadius,GridMaxRadius) /
grid_scale_delta_lon_mpmas;

/* get signal amplitude for each robot */
        amplitude = SignalOne( lat, lon );

/* update the robot's struct */
        Update_VehicleGA(vehicle, lat, lon, amplitude);

                printf( "%6d %3d %5d %10ld %10ld %10ld %10ld %10.5f %8.4f %8.4f \n",
                                0, vehicle, RoBug[vehicle].RoBugDIR, lat, lon,
                                target_location_lat, target_location_lon,
                                RoBug[vehicle].RoBugSIG,
                        -(lon - target_location_lon)*grid_scale_delta_lon_mpmas,
                                (lat - target_location_lat)*grid_scale_delta_lat_mpmas
                                );
                fprintf( OutputFP,"%6d %3d %5d %10ld %10ld %10ld %10ld %10.5f %8.4f %8.4f \n",
                                0, vehicle, RoBug[vehicle].RoBugDIR, lat, lon,
                                target_location_lat, target_location_lon,
                                RoBug[vehicle].RoBugSIG,
                                -(lon - target_location_lon)*grid_scale_delta_lon_mpmas,
                                (lat - target_location_lat)*grid_scale_delta_lat_mpmas
                                );
                }
        }

/* For this vehicle */
        lat=target_location_lat + RandomLong(-GridMaxRadius,GridMaxRadius) /
grid_scale_delta_lat_mpmas;
        lon=target_location_lon + RandomLong(-GridMaxRadius,GridMaxRadius) /
grid_scale_delta_lon_mpmas;

        RoBug[this_vehicle_ID].RoBugXPOS = lon / delta_lon;
        RoBug[this_vehicle_ID].RoBugYPOS = lat / delta_lat;

/* print heading for output */
        printf("\n \n >> Coordinate time history of vehicle # %d \n",this_vehicle_ID);
        printf("          [1 is where the robot WAS; 2 is current location]\n");
        printf(" Time Nbr1 Nbr2 Dir  Lat1       Lon1       Lat2       Lon2"
                "    TargetLat  TargetLon  Signl  Xdist(m) Ydist(m)\n");

        fprintf(OutputFP,"\n \n >> Coordinate time history of vehicle # %d \n",this_vehicle_ID);
        fprintf(OutputFP," 1 is where the robot WAS; 2 is current location\n");
        fprintf(OutputFP," Time Nbr1 Nbr2 Dir  Lat1       Lon1       Lat2       Lon2"
                "    TargetLat TargetLon  Signl  Xdist(m) Ydist(m)\n");

/* time loop */
        for (timestep=0; timestep<10000; timestep++)
        {
                time=(float)timestep/iterations_per_second;

                amplitude = SignalOne( lat, lon );

                UpdateGA( &dlat, &dlon, this_vehicle_ID, lat, lon, amplitude,
                        &neighbor1_ID, &neighbor2_ID);
```

```
			printf( "%5.2f %3d %3d %3d %10ld %10ld %10ld %10ld %10ld %10ld %6.1f %8.4f
%8.4f \n",
				time, neighbor1_ID, neighbor2_ID, RoBug[this_vehicle_ID].RoBugDIR,
				lat, lon, dlat, dlon,
				target_location_lat, target_location_lon,
				RoBug[this_vehicle_ID].RoBugSIG,
				-(dlon - target_location_lon)*grid_scale_delta_lon_mpmas,
				(dlat - target_location_lat)*grid_scale_delta_lat_mpmas
				);

			fprintf( OutputFP,"%5.2f %3d %3d %3d %10ld %10ld %10ld %10ld %10ld %10ld
%6.1f %8.4f %8.4f \n",
				time, neighbor1_ID, neighbor2_ID,RoBug[this_vehicle_ID].RoBugDIR,
				lat, lon, dlat, dlon,
				target_location_lat, target_location_lon,
				RoBug[this_vehicle_ID].RoBugSIG,
				-(dlon - target_location_lon)*grid_scale_delta_lon_mpmas,
				(dlat - target_location_lat)*grid_scale_delta_lat_mpmas
					);

/* Assume we ge there immediately */
			lat = dlat;
			lon = dlon;

			if(timestep == total_iterations) {
				printf("END\n");
				fprintf(OutputFP,"END");
				exit(1);
			}

		}

/*		UpdateGA(&dlat, &dlon, vehicle, RoBugLocal[vehicle].lat, RoBugLocal[vehicle].lon,
RoBugLocal[vehicle].amplitude );
*/
		KillGA(	);  // Note change ***
}


/***************************** RandomDouble ****************/

double  RandomDouble ( double start, double stop )
{

/*   return (start+(stop-start)*rand()/32767.0); */

/* dwb */
   return (start+(stop-start)*randomng(&iseed));

}


/***************************** RandomLong ******************/

long  RandomLong (long istart, long istop)
{

   int i;
```

```
   long delta;
           double random;
   i=0;
   do {
/*     delta=(istop-istart+1) * (rand()/32767.0); */
       i++;

       if(i>100) {
          printf(" %%%%% FUNCTION NOT FINDING RANDOM NUMBER BETWEEN LIMITS! \n");
          printf("   In RandomLong, i = %d \n",i);
          printf(" >>>>  RL: istart, istop, delta = %d %d %d \n",
             istart,istop,delta);
          printf(" >>>>       iseed, &iseed = %d, %d \n",iseed, &iseed);
          exit(1);
       }


/* dwb */
           random=randomng(&iseed);
           delta=(istop-istart+1) * random;

   }  while (istart+delta < istart || istart+delta > istop);

   return (istart+delta);

}


/**************** random number generator ******************/

double randomng(int *pp)
{

/*
   returns a value between 0 and 1
       mm      =       length of unsigned long integer
       aa      =       number to ensure good random number generation
*/

   double aa = 16807.0;
   double mm = 2147483647.0;
/*      double sseed; */
/*      int iseed; */

   sseed=*pp;

/*      printf(" >>  pp = %d \n", pp);
       printf(" >> *pp = %d \n",*pp);
       fflush(stdout);
*/

   sseed=fmod(aa*sseed,mm);
   iseed=sseed;
/*      RandomPointer = &iseed; */
/*      printf(" >> iseed = %d \n",iseed); */
/*
       printf(" >>?? sseed, mm = %f, %f \n",sseed,mm);
```

```
        printf(" >>?? iseed = %d \n",iseed);
        fflush(stdout);
*/

/* for double random(int *pp) */
   return sseed/mm;


/* for int random(int *pp) */
/*      return iseed; */

}


/***************************** SignalOne ****************/
int SignalOne( long lat, long lon ) {

            double signal_strength, rsqr, rsqrMax;
            int signal;
            float xdiff, ydiff;

/* Calculate signal strength for all robots */

            xdiff = (lon - target_location_lon) * grid_scale_delta_lon_mpmas ;
            ydiff = (lat - target_location_lat) * grid_scale_delta_lat_mpmas ;

            rsqr=xdiff*xdiff+ydiff*ydiff;


/*     if (rsqr > rsqrMax) rsqrMax=rsqr; */

/*----------------------------------------------------------------------------*/
/* 1. Signal strength: 1/(r**2) */
/* Signal strength needs to vary between 0 and 1 */

/*   original signal in GA code */
/*          signal_strength=1.0/(rsqr + 1.); */

/* Signal strength appropriate for this code */
/* Fit 1/r**2 equation for 4095 when r=1 meter and 50 when r=50 meters */
/*          signal_strength=(rsqrMax - rsqr)/rsqrMax; */             /* values between 0 and 1 */
/*          signal=SignalStrengthMax * signal_strength; */ /* values between 0 and SignalStrengthMax */

signal=4045.8092 * (1. / rsqr)+49.1908;
signal=4095*(-0.5*sqrt(rsqr)/50+1.);
if(signal>4095) signal=4095;

            if(signal < 0 ) {
                        printf("\n\n Signal is less than zero in func. SignalOne\n");
                        printf(" signal = %d\n",signal);
                        exit(1);
            }

   return signal;

}

/***************************** InitGA ****************/
```

```
void InitGA( )
{
	int	i;


/* dynamically allocate storage to Robug for all vehicles */

/*  (The value of RoBug is a pointer to the allocated memory) */
	RoBug = (RoBugType *)malloc( numvehicles * sizeof( RoBugType ) );

/* Memory check for malloc */
	if(!RoBug)
	{
		printf("\n Memory allocation error; program halted \n");
		exit(1);
	}

/* initialize bug position parameters */
/* Zero everything in struct but ID and DIR */

      for (i=0; i < numvehicles; i++)
		{
/* id the vehicle */
	RoBug[i].RoBugID=i;

/* initial heading where rattlers need to go */
	RoBug[i].RoBugDIR=RandomLong(1,4);
/*	printf( " RoBug[i].RoBugDIR = %d \n",RoBug[i].RoBugDIR);
*/

/* Random placement of bugs over grid, spaced randomly within -GridMaxRadius
		to +GridMaxRadius meters of each other */
	RoBug[i].RoBugXPOS=0;
	RoBug[i].RoBugYPOS=0;

/* signal received by robug */
	RoBug[i].RoBug1SIG=0;

/* wall position */
	RoBug[i].RoBugXVERT=0;
	RoBug[i].RoBugYVERT=0;
	RoBug[i].RoBugXHORZ=0;
	RoBug[i].RoBugYHORZ=0;

/* nearest neighbors */
	RoBug[i].RoBug1XPOS=0;
	RoBug[i].RoBug1YPOS=0;
	RoBug[i].RoBug1SIG=0;
	RoBug[i].RoBug2XPOS=0;
	RoBug[i].RoBug2YPOS=0;
	RoBug[i].RoBug2SIG=0;

/* zero the registers */
	RoBug[i].aRegister=0.0;
	RoBug[i].bRegister=0.0;
	RoBug[i].cRegister=0.0;
	RoBug[i].Average_X_Register=0.0;
	RoBug[i].Average_Y_Register=0.0;
```

```
        }

}


/***************************** UpdateGA ****************/
void UpdateGA(

/* desired outputs */
/* units: dlat, dlon: milliarcseconds */
long *dlat, long *dlon,

/* inputs */
/* current position, orientation, signal strength for 'this_vehicle' */
/* amplitude varies 0-4095 */
int id,
long lat, long lon, int amplitude,

/*  id of the two nearest neighbors with which the main vehicle communicates */
int *neighbor1, int *neighbor2 )
{
            int         i, j, jmin1, jmin2;
            double rmin, rsqr;
            long xdiff, ydiff;

/* scale by delta_lon to get XPOS into grid units; needed for MOVEGA where an
     increment of 1 implies one grid unit */
/*
            RoBug[id].RoBugXPOS = lon / delta_lon;
            RoBug[id].RoBugYPOS = lat / delta_lat;
*/
            RoBug[id].RoBugSIG =  (double) amplitude ; /* / SignalStrengthMax*/


/* Assume no wall ahead, WallAhead=0 (false); if wall, WallAhead=1 (true) */
            RoBug[id].WallAhead = 0;

/* Calculate nearest neighbors */

          /* update bug neighbors only for 'this_vehicle_ID' for this simulation */

/*          for(i=0; i<numvehicles; i++) { */
            i=id;
             jmin1=i;
             jmin2=i;

             /*  find 1st nearest bug */
             rmin=REALLYBIGNUMBER;
             for (j=0; j < numvehicles; j++)
                                             {
                if (j != i)
                                                        {
                   xdiff=RoBug[i].RoBugXPOS - RoBug[j].RoBugXPOS;
                   ydiff=RoBug[i].RoBugYPOS - RoBug[j].RoBugYPOS;
                   rsqr=xdiff*xdiff+ydiff*ydiff;
                   if (rsqr < rmin)
                                                                  {
                      jmin1=j;
```

```
                rmin=rsqr;
              }
            }
          }

          RoBug[i].RoBug1XPOS = RoBug[jmin1].RoBugXPOS;
          RoBug[i].RoBug1YPOS = RoBug[jmin1].RoBugYPOS;
          RoBug[i].RoBug1SIG  = RoBug[jmin1].RoBugSIG;
                                    *neighbor1=jmin1;

          /*  find 2nd nearest bug */
          rmin=REALLYBIGNUMBER;
          for (j=0; j < numvehicles; j++)
                                    {
            if (j != i && j != jmin1)
                                            {
              xdiff=RoBug[i].RoBugXPOS-RoBug[j].RoBugXPOS;
              ydiff=RoBug[i].RoBugYPOS-RoBug[j].RoBugYPOS;
              rsqr=xdiff*xdiff+ydiff*ydiff;
              if (rsqr < rmin)
                                                    {
                jmin2=j;
                rmin=rsqr;
              }
            }
          }
          RoBug[i].RoBug2XPOS=RoBug[jmin2].RoBugXPOS;
          RoBug[i].RoBug2YPOS=RoBug[jmin2].RoBugYPOS;
          RoBug[i].RoBug2SIG=RoBug[jmin2].RoBugSIG;
          *neighbor2=jmin2;
/*                                  } */

/* call the routine generated by CEDAR */
         MoveGA(&RoBug[id]);

         *dlon =  (double) RoBug[id].RoBugXPOS * delta_lon;
         *dlat =  (double) RoBug[id].RoBugYPOS * delta_lat;

}

/****************************** KillGA ****************/
void KillGA()
{
  free ( (char *)RoBug );
}


/****************************** Update_VehicleGA ****************/
void Update_VehicleGA(int id, long lat, long lon, int amplitude)
{
/* update other vehicles */
/* nondimensionalize XPOS and YPOS to grid spacings  for GA algorithm*/
/* unscale SIG for GA algorithm */
         RoBug[id].RoBugXPOS = lon / delta_lon;
         RoBug[id].RoBugYPOS = lat / delta_lat;
         RoBug[id].RoBugSIG =  (double)amplitude ; /* / SignalStrengthMax; */

/* Assume no wall ahead, WallAhead=0 (false); if wall, WallAhead=1 (true) */
```

```
            RoBug[id].WallAhead = 0;

}


/****************************** MoveAhead **********************/
double MoveAhead(RoBugType *bug) {
            double value;

          if (!bug->WallAhead) {
                         switch (bug->RoBugDIR) {
                                 case 1: /* move north */
                                             bug->RoBugYPOS++;
                                             break;
                                 case 2: /* move east */
                                             bug->RoBugXPOS++;
                                             break;
                                 case 3: /* move south */
                                             bug->RoBugYPOS--;
                                             break;
                                 case 4: /* move west */
                                             bug->RoBugXPOS--;
                                             break;
                                 default:
                                             exit(1);
                                             break;
                                 }
             value = 1.0;
          }
          else {
             value = -1.0;
          }

            return value;
}


/****************************** TurnRight **********************/
double TurnRight(RoBugType *bug) {
            long direct;
            double value;

               direct=bug->RoBugDIR;
      direct++;
      if (direct == 5) direct=1;
      bug->RoBugDIR = direct;

                      value=1.;


  return value;
}




/****************************** TurnNorth **********************/
double TurnNorth(RoBugType *bug) {
```

```
        double value;

   bug->RoBugDIR = 1;
/*      ReturnFlag=1; */
        value=1;

        return value;
}


/***************************** TurnEast ***********************/
double TurnEast(RoBugType *bug) {
        double value;

   bug->RoBugDIR = 2;
/*      ReturnFlag=1; */
        value=1;

        return value;
}


/***************************** TurnSouth ***********************/
double TurnSouth(RoBugType *bug) {
        double value;

   bug->RoBugDIR = 3;
/*      ReturnFlag=1; */
        value=1;

        return value;
}


/***************************** TurnWest ***********************/
double TurnWest(RoBugType *bug) {
        double value;

   bug->RoBugDIR = 4;
/*      ReturnFlag=1; */
        value=1;

        return value;
}


/***************************** MoveGA ****************/

double MoveGA(RoBugType *bug)  {

/* >> INSERT GENETIC ALGORITHM HERE << */

}

/* THE END */
```

Appendix C
ROBOCOP.C Sample Output Listing for Model 0

```
total iterations = 50
iterations per second = 5
number of vehicles = 5
grid_spacing_lat_m = 3
grid_spacing_lon_m = 3
grid_scale_delta_lat_mpmas = 0.031044
grid_scale_delta_lon_mpmas = 0.025430
delta_lat (mas/lat_grid_step) = 96.635648
delta_lon (mas/lon_grid_step) = 117.969560


>> Locations of all vehicles except vehicle # 3
 Time  Veh   Dir     Lat        Lon    Target_lat Target_lon    Signal  X dist(m) Y dist(m)
    0   0     1  126000012  921601923  126000000  921600000 2092.00000 -48.9024   0.3725
    0   1     3  125999457  921600636  126000000  921600000 3138.00000 -16.1737 -16.8571
    0   2     1  125998518  921599524  126000000  921600000 2146.00000  12.1048 -46.0079
    0   4     3  125999528  921600741  126000000  921600000 3117.00000 -18.8438 -14.6530


>> Coordinate time history of vehicle # 3
1 is where the robot WAS; 2 is current location
Time Nbr1 Nbr2 Dir  Lat1       Lon1       Lat2       Lon2       TargetLat TargetLon  Signl  Xdist(m) Ydist(m)
0.00   0    4   4  125999838  921601572  125999742  921601441  126000000  921600000 2445.0 -36.6450  -8.0095
0.20   0    4   4  125999742  921601441  125999742  921601323  126000000  921600000 2558.0 -33.6443  -8.0095
0.40   0    4   4  125999742  921601323  125999742  921601205  126000000  921600000 2678.0 -30.6435  -8.0095
0.60   4    1   4  125999742  921601205  125999742  921601087  126000000  921600000 2797.0 -27.6427  -8.0095
0.80   4    1   4  125999742  921601087  125999742  921600969  126000000  921600000 2916.0 -24.6420  -8.0095
1.00   4    1   4  125999742  921600969  125999742  921600851  126000000  921600000 3033.0 -21.6412  -8.0095
1.20   4    1   4  125999742  921600851  125999742  921600733  126000000  921600000 3150.0 -18.6404  -8.0095
1.40   4    1   4  125999742  921600733  125999742  921600615  126000000  921600000 3264.0 -15.6396  -8.0095
1.60   1    4   4  125999742  921600615  125999742  921600497  126000000  921600000 3375.0 -12.6389  -8.0095
1.80   1    4   4  125999742  921600497  125999742  921600379  126000000  921600000 3482.0  -9.6381  -8.0095
2.00   1    4   4  125999742  921600379  125999742  921600261  126000000  921600000 3581.0  -6.6373  -8.0095
2.20   1    4   4  125999742  921600261  125999742  921600143  126000000  921600000 3669.0  -3.6365  -8.0095
2.40   1    4   4  125999742  921600143  125999742  921600025  126000000  921600000 3734.0  -0.6358  -8.0095
2.60   1    4   4  125999742  921600025  125999742  921599907  126000000  921600000 3765.0   2.3650  -8.0095
2.80   1    4   1  125999742  921599907  125999742  921599907  126000000  921600000 3753.0   2.3650  -8.0095
3.00   1    4   1  125999742  921599907  125999839  921599907  126000000  921600000 3753.0   2.3650  -4.9982
3.20   1    4   1  125999839  921599907  125999935  921599907  126000000  921600000 3868.0   2.3650  -2.0179
3.40   1    4   1  125999935  921599907  126000032  921599907  126000000  921600000 3967.0   2.3650   0.9934
3.60   1    4   1  126000032  921599907  126000129  921599907  126000000  921600000 3989.0   2.3650   4.0047
3.80   1    4   2  126000129  921599907  126000129  921599907  126000000  921600000 3904.0   2.3650   4.0047
4.00   1    4   2  126000129  921599907  126000129  921600025  126000000  921600000 3904.0  -0.6358   4.0047
4.20   1    4   2  126000129  921600025  126000129  921600143  126000000  921600000 3928.0  -3.6365   4.0047
4.40   1    4   3  126000129  921600143  126000129  921600143  126000000  921600000 3873.0  -3.6365   4.0047
4.60   1    4   3  126000129  921600143  126000032  921600143  126000000  921600000 3873.0  -3.6365   0.9934
4.80   1    4   3  126000032  921600143  125999935  921600143  126000000  921600000 3940.0  -3.6365  -2.0179
5.00   1    4   4  125999935  921600143  125999935  921600143  126000000  921600000 3924.0  -3.6365  -2.0179
5.20   1    4   4  125999935  921600143  125999935  921600025  126000000  921600000 3924.0  -0.6358  -2.0179
5.40   1    4   4  125999935  921600025  125999935  921599907  126000000  921600000 4008.0   2.3650  -2.0179
5.60   1    4   1  125999935  921599907  125999935  921599907  126000000  921600000 3967.0   2.3650  -2.0179
5.80   1    4   1  125999935  921599907  126000032  921599907  126000000  921600000 3967.0   2.3650   0.9934
6.00   1    4   1  126000032  921599907  126000129  921599907  126000000  921600000 3989.0   2.3650   4.0047
6.20   1    4   2  126000129  921599907  126000129  921599907  126000000  921600000 3904.0   2.3650   4.0047
6.40   1    4   2  126000129  921599907  126000129  921600025  126000000  921600000 3904.0  -0.6358   4.0047
6.60   1    4   2  126000129  921600025  126000129  921600143  126000000  921600000 3928.0  -3.6365   4.0047
6.80   1    4   3  126000129  921600143  126000129  921600143  126000000  921600000 3873.0  -3.6365   4.0047
7.00   1    4   3  126000129  921600143  126000032  921600143  126000000  921600000 3873.0  -3.6365   0.9934
7.20   1    4   3  126000032  921600143  125999935  921600143  126000000  921600000 3940.0  -3.6365  -2.0179
7.40   1    4   4  125999935  921600143  125999935  921600143  126000000  921600000 3924.0  -3.6365  -2.0179
7.60   1    4   4  125999935  921600143  125999935  921600025  126000000  921600000 3924.0  -0.6358  -2.0179
7.80   1    4   4  125999935  921600025  125999935  921599907  126000000  921600000 4008.0   2.3650  -2.0179
8.00   1    4   1  125999935  921599907  125999935  921599907  126000000  921600000 3967.0   2.3650  -2.0179
8.20   1    4   1  125999935  921599907  126000032  921599907  126000000  921600000 3967.0   2.3650   0.9934
8.40   1    4   1  126000032  921599907  126000129  921599907  126000000  921600000 3989.0   2.3650   4.0047
8.60   1    4   2  126000129  921599907  126000129  921599907  126000000  921600000 3904.0   2.3650   4.0047
8.80   1    4   2  126000129  921599907  126000129  921600025  126000000  921600000 3904.0  -0.6358   4.0047
9.00   1    4   2  126000129  921600025  126000129  921600143  126000000  921600000 3928.0  -3.6365   4.0047
9.20   1    4   3  126000129  921600143  126000129  921600143  126000000  921600000 3873.0  -3.6365   4.0047
9.40   1    4   3  126000129  921600143  126000032  921600143  126000000  921600000 3873.0  -3.6365   0.9934
9.60   1    4   3  126000032  921600143  125999935  921600143  126000000  921600000 3940.0  -3.6365  -2.0179
9.80   1    4   4  125999935  921600143  125999935  921600143  126000000  921600000 3924.0  -3.6365  -2.0179
10.00   1    4   4  125999935  921600143  125999935  921600025  126000000  921600000 3924.0  -0.6358  -2.0179
END
```

Appendix D
Mathematica 4.0 Graphics Program Listing
(as used for Model 0)

```
Clear[]

(* Author: Daniel W. Barnette, Sandia National Laboratories
*)

(* The list of data, DataList, extracted from ROBOCOP.C output, contains the following items :

          Vehicle                                     OtherVehicles
============================             =================
Column    Data                               Data
----------  ------                               ------
    1     Time                                 Time
    2     Neighbor1 (first closest)            Vehicle No.
    3     Neighbor2 (second closest)           Direction
    4     Direction robot is facing            Latitude
    5     Latitude (milliarcseconds)           Longitude
    6     Longitude (milliarcseconds)          Target Latitude
    7     Latitude (milliarcseconds)           Target Longitude
    8     Longitude (milliarcseconds)          Signal strength
    9     Target Latitude (milliarcseconds)    X distance from target
   10     Target Longitude (milliarcseconds)   Y distance from target
   11     Signal (varies from 0 to 4095)
   12     X distance from target (meters)
   13     Y distance from target (meters)
 *)

Clear[
 OtherVehiclesPlot,
 VehiclePlot,
 DataFile,
 DataWord,
 Description1, Description2,
 TodaysDateAndTime,
 RowsColumns,
 NumVehicles,
 TotalIterations,
 IterationsPerSecond
 ]

Date[]

Description1 = "Genetic Algorithms Simulator";

Description2 = "Signal Source: 1/r**2";

TodaysDateAndTime := (
   Temp = Date[];
   StringForm[
    "``   Date: ``/``/``    Time: ``:``:``",
    Description1,
```

```
    Temp[[2]], Temp[[3]], Temp[[1]],
    Temp[[4]], Temp[[5]], Temp[[6]]
    ]
   )

(* Uncomment following to check if file can be opened; for debugging code *)

(* ! ! "d:\Program \
Files\DevStudio\MyProjects\Robug_Simulator\RobocopOutputMod0.txt" *)

DataFile =
  OpenRead["d:\Program \
Files\DevStudio\MyProjects\Robug_Simulator\RobocopOutputMod0.txt"]

DataWord = "NULL";

While[
  DataWord != "=" ,
  DataWord = Read[DataFile, Word];
  ]

TotalIterations = Read[DataFile, Number]

DataWord = "NULL";

While[
  DataWord != "=" ,
  DataWord = Read[DataFile, Word];
  ]

IterationsPerSecond = Read[DataFile, Number]

DataWord = "NULL";

While[
  DataWord != "=" ,
  DataWord = Read[DataFile, Word];
  ]

NumVehicles = Read[DataFile, Number]

While[
  DataWord != "Time",
  Skip[DataFile, Record];
  DataWord = Read[DataFile, Word];
  ]

Skip[DataFile, Record]

OtherVehiclesPlot =
   Table[Read[DataFile, Number], {NumOfDataLines,
      NumVehicles - 1}, {NumOfDataColumns, 10}];


TableForm[OtherVehiclesPlot]
```

```
Do[
  OtherVehiclesPlot[[i]] = Append[OtherVehiclesPlot[[i]], 0],
  {i, NumVehicles - 1}
  ]

Table[Dimensions[OtherVehiclesPlot]]

TableForm[OtherVehiclesPlot]

While[
  DataWord != "Dir",
  DataWord = Read[DataFile, Word];
  ]

Skip[DataFile, Record]

VehiclePlot =
    Table[
      Read[DataFile, Number], {NumOfDataLines,
        TotalIterations + 1}, {NumOfDataColumns, 13}];
RowsColumns = Table[Dimensions[VehiclePlot]];

Close[DataFile]

RowsColumns

Do[
  VehiclePlot[[i]] = Append[VehiclePlot[[i]], 0],
  {i, TotalIterations + 1}
  ]

TableForm[VehiclePlot]

(*
  Get Graphics packages needed for plots
  *)

<< Graphics`Graphics3D`

<< Graphics`Arrow`

(* << Graphics`Polyhedra` *)

<< Graphics`Animation`

SignalMax = 200

Signal[x_, y_] := (signal = (4045.8092*(1./(x*x + y*y + 0.0001)) + 49.1908);
    If[signal > SignalMax, SignalMax, signal] )

signalTable = Table[{x, y, Signal[x, y]}, {x, -50, 50, 2}, {y, -50, 50, 2}];

signalPlot =
  ListSurfacePlot3D[signalTable, PlotRange -> {0, 200}, Axes -> True,
    ColorFunction -> Hue, ImageSize -> 500,
    BoxRatios -> {1.1, 1.1, 1} (* DisplayFunction -> Identity*)]
```

```
signalContour =
  ContourPlot[Signal[x, y], {x, -50, 50}, {y, -50, 50}, PlotPoints -> 25,
    ColorFunction -> GrayLevel, ContourLines -> True, Contours -> 10,
    ContourShading -> False, ImageSize -> 500]

Clear[signalShadow]

signalShadow =
    ShadowPlot3D[Signal[x, y] - 77, {x, -50, 50}, {y, -50, 50},
      PlotPoints -> 40, ShadowMesh -> False, Axes -> True,
      AxesLabel -> {X, Y, Signal},  ImageSize -> 600, ShadowPosition -> 1,
      SurfaceMesh -> True, ViewPoint -> {1.464, -2.702, 1.417}
      ];

(*
  SpinShow[signalShadow, Frames -> 30, ViewPoint -> {1.464, -2.702, 1.417},
    SpinTilt -> {0, 0}, SpinDistance -> 5, Axes -> False, ImageSize -> 600
    ]
  *)

(*
  plotTet = Polyhedron[Tetrahedron, {0, 0, SignalMax}, 2, Boxed -> True,
       ImageSize -> 400,
      PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50, SignalMax + 50}},
      Axes -> True, FaceGrids -> {{0, 0, -1}}
      ]
  *)

Clear[plot00, plot01, plot10, plot11, plot421, plot521]

(* Initial Bug Location *)

plot00 =
  ScatterPlot3D[
    {
      {
        VehiclePlot[[1, 12]],
        VehiclePlot[[1, 13]],
        VehiclePlot[[1, 14]] +
          Signal[VehiclePlot[[1, 12]], VehiclePlot[[1, 13]]] - 72
        },
      {
        VehiclePlot[[1, 12]],
        VehiclePlot[[1, 13]],
        VehiclePlot[[1, 14]] +
          Signal[VehiclePlot[[1, 12]], VehiclePlot[[1, 13]]] - 72
        }
      },
    PlotRange -> {{-50, 50}, {-50, 50}, {-50, +50}},
    PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005]} ,
    DisplayFunction -> Identity
    ]


plot01 =
```

```
 ScatterPlot3D[
  {
   {VehiclePlot[[1, 12]],
    VehiclePlot[[1, 13]],
    VehiclePlot[[1, 14]] + SignalMax
    },
   {
    VehiclePlot[[1, 12]],
    VehiclePlot[[1, 13]],
    VehiclePlot[[1, 14]] + SignalMax}
   },
  PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50, SignalMax + 50}},
  PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005]} ,
  DisplayFunction -> Identity
  ]

(* Other Bug Locations *)

plot10 = ScatterPlot3D[
  Table[
   {
    OtherVehiclesPlot[[i, 9]],
    OtherVehiclesPlot[[i, 10]],
    OtherVehiclesPlot[[i, 11]] +
     Signal[OtherVehiclesPlot[[i, 9]], OtherVehiclesPlot[[i, 10]]] - 72
    },
   {i, NumVehicles - 1}
   ] ,
   PlotStyle -> {Hue[0.6], PointSize[0.02]}, PlotJoined -> False,
  PlotRange -> {{-50, 50}, {-50, 50}, {-50, 50}},
  DisplayFunction -> Identity
  ]

plot11 = ScatterPlot3D[
  Table[{
    OtherVehiclesPlot[[i, 9]],
    OtherVehiclesPlot[[i, 10]],
    OtherVehiclesPlot[[i, 11]] + SignalMax
    },
   {i, NumVehicles - 1}
   ] ,
  PlotStyle -> {Hue[0.6], PointSize[0.02]}, PlotJoined -> False,
  PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50, SignalMax + 50}},
  DisplayFunction -> Identity
  ]

plot421 = ScatterPlot3D[
  {
   {
    VehiclePlot[[1, 12]],
    VehiclePlot[[1, 13]],
    VehiclePlot[[1, 14]] + SignalMax
    },
   {
    OtherVehiclesPlot[[1, 9]],
    OtherVehiclesPlot[[1, 10]],
```

```
    OtherVehiclesPlot[[1, 11]] + SignalMax
    }
   },
  PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50, SignalMax + 50}},
  PlotJoined -> True,
  PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005],
    Dashing[{0.002, 0.008, 0.002, 0.008}]} , DisplayFunction -> Identity
  ]


plot521 = ScatterPlot3D[
  {
   {
    VehiclePlot[[1, 12]],
    VehiclePlot[[1, 13]],
    VehiclePlot[[1, 14]] + SignalMax
    },
   {
    OtherVehiclesPlot[[4, 9]],
    OtherVehiclesPlot[[4, 10]],
    OtherVehiclesPlot[[4, 11]] + SignalMax
    }
   },
  PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50, SignalMax + 50}},
  PlotJoined -> True,
  PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005],
    Dashing[{0.01, 0.01}]} , DisplayFunction -> Identity
  ]


signalSpin =
 Show[signalShadow, plot00, plot01, plot10, plot11, plot421, plot521,
  DisplayFunction -> $DisplayFunction, ViewPoint -> {1.384, -2.555, 1.734},
  ImageSize -> 600,
  PlotLabel -> StyleForm[  "Initial Conditions", "Section"]]

SpinShow[signalSpin, Frames -> 30, ViewPoint -> {1.384, -2.555, 1.734},
 SpinTilt -> {0, 0}, SpinDistance -> 5, Axes -> False, ImageSize -> 600
 ]

Clear[plot100, plot101, plot200, plot300, plot301, plot400, plot401]

gr = Do[
  (
   plot100 = ScatterPlot3D[
     Table[{
       VehiclePlot[[i, 12]],
       VehiclePlot[[i, 13]],
       VehiclePlot[[i, 14]] + SignalMax
        },
      {i, j, j}] ,
     PlotJoined -> False,
     PlotStyle -> { PointSize[0.02], Thickness[0.005], Hue[0.6]},
     PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
         SignalMax + 50}} , DisplayFunction -> Identity
     ];
```

```
plot101 = ScatterPlot3D[
   Table[{
      VehiclePlot[[i, 12]],
      VehiclePlot[[i, 13]],

      VehiclePlot[[i, 14]] +
        Signal[VehiclePlot[[i, 12]], VehiclePlot[[i, 13]]] - 72
       },
     {i, j, j}] ,
   PlotJoined -> False,
   PlotStyle -> { PointSize[0.02], Thickness[0.005], Hue[0.6]},
   PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
        SignalMax + 50}} , DisplayFunction -> Identity
   ];

plot300 = ScatterPlot3D[
   Table[{
      VehiclePlot[[i, 12]],
      VehiclePlot[[i, 13]],
      VehiclePlot[[i, 14]] + SignalMax
       },
     {i, j}],
   PlotJoined -> True,
   PlotStyle -> { PointSize[0.5], Thickness[0.008], Hue[0.17]},
   PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
        SignalMax + 50}}, DisplayFunction -> Identity
   ];

plot301 = ScatterPlot3D[
   Table[{
      VehiclePlot[[i, 12]],
      VehiclePlot[[i, 13]],

      VehiclePlot[[i, 14]] +
        Signal[VehiclePlot[[i, 12]], VehiclePlot[[i, 13]] ] - 72
      },
     {i, j}] ,
   PlotJoined -> True,
   PlotStyle -> { PointSize[0.5], Thickness[0.008], Hue[0.17]},
   PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
        SignalMax + 50}} , DisplayFunction -> Identity
   ];

(* Show nearest neighbors with connecting lines *)

For[k = 1, k < NumVehicles, k++,
  If [
   VehiclePlot[[j, 2]] == OtherVehiclesPlot[[k, 2]],
   plot400 = ScatterPlot3D[
      {
        {
         VehiclePlot[[j, 12]],
         VehiclePlot[[j, 13]],
         VehiclePlot[[j, 14]] + SignalMax
         },
```

```
        {
          OtherVehiclesPlot[[k, 9]],
          OtherVehiclesPlot[[k, 10]],
          OtherVehiclesPlot[[k, 11]] + SignalMax
          }
        },

      PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
            SignalMax + 50}}, PlotJoined -> True,

      PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005],
          Dashing[{0.002, 0.008, 0.002, 0.008}]} ,
      DisplayFunction -> Identity
      ]
    ];

   If [
    VehiclePlot[[j, 3]] == OtherVehiclesPlot[[k, 2]],
    plot401 = ScatterPlot3D[
      {
        {
          VehiclePlot[[j, 12]],
          VehiclePlot[[j, 13]],
          VehiclePlot[[j, 14]] + SignalMax
          },
        {
          OtherVehiclesPlot[[k, 9]],
          OtherVehiclesPlot[[k, 10]],
          OtherVehiclesPlot[[k, 11]] + SignalMax
          }
        },

      PlotRange -> {{-50, 50}, {-50, 50}, {SignalMax - 50,
            SignalMax + 50}}, PlotJoined -> True,

      PlotStyle -> {GrayLevel[0.], PointSize[0.02], Thickness[0.005],
          Dashing[{0.01, 0.01}]} , DisplayFunction -> Identity
      ]
    ];
   ];

  Show[
   signalShadow, plot00, plot01, plot10, plot11, plot100, plot101,
   plot300, plot301, plot400, plot401,
   DisplayFunction -> $DisplayFunction,
   ViewPoint -> {1.433, -2.646, 1.548},
   PlotLabel ->
    StyleForm[  N[j/IterationsPerSecond]  " seconds", "Section"],
   ImageSize -> 600
   ]

  ),
 {j, RowsColumns[[1]]}
 ]

(* The End *)
```

EXTERNAL DISTRIBUTION:

Copies:	Name/Entity:
4	Santa Fe Institute Attn: Melanie Mitchell (2 copies) James Crutchfield (2 copies) 1399 Hyde Park Road Santa Fe, New Mexico 87501

INTERNAL DISTRIBUTION:

Copies:	Mail Stop:	Name/Org:
1	1002	P. Eicker, 15200
1	0316	S. S. Dosanjh, 9233
1	0318	G. S. Davidson, 9212
5	0318	R. J. Pryor, 9212
1	0318	M. Boslough, 9212
1	0318	K. Boyack, 9212
1	0318	R. Hightower, 9212
1	0825	W. H. Rutledge, 9115
1	0321	W. J. Camp, 9200
10	0316	D. W. Barnette, 9233
1	1110	D. E. Womble, 9214
1	0321	A. L. Hale, 9220
1	0316	G. S. Heffelfinger, 9235
1	0847	R. W. Leland, 9226
1	0819	E. Boucheron, 9231
1	0310	P. Yarrington, 9230
1	0847	D. R. Martinez, 9124
1	1003	R. Robinett, 15211
5	1003	J. Hurtado, 15211
1	1003	J. Feddema, 15211
1	1003	C. Lewis, 15211
1	1004	D. Schoenwald, 15221
1	1010	G. R. Eisler, 15222
1	0839	G. Yonas, 16000
1	9018	Central Technical Files, 8945-1
2	0899	Technical Library, 9616
1	0612	Review & Approval Desk, 9612 For DOE/OSTI